工学選書 2

68000 マシン語プログラミング

第一システム　村山仁郎　著

工学社

工学選書 2

68000 マシン語プログラミング

第一システム　村山仁郎　著

工学社

はじめに

68000マイコンは，一般には16ビット・マイコンに分類されていますが，その内部レジスタは32ビット長で構成されており，16ビット/32ビット・マイコンとも呼ぶべきアーキテクチャを持っています．8088（内部レジスタは8086と同じで，外部バスが8ビット幅）を16ビット・マイコンに分類するのであれば，68000は32ビット・マイコンに分類してもおかしくはありません．しかし，一般には16ビット・マイコンに分類されているようですので，ここでも16ビット・マイコンとして扱います．

チップ集積度，内部レジスタ幅，アドレス空間の大きさを68000と8086とで比較してみると，下表のようになります．

68000はかなり高性能な16ビット・マイコンで，32ビット・マイコンを意識して設計されているともいえます．

命令体系も強力で，いろいろな種類の命令がバランスよく配備されており，種々のアドレッシング・モードを用いることができるようになっています．ビット・データから32ビットのロング・ワード・データまで扱うことができ，この点が他の16ビット・マイコンにくらべて68000の大きな特徴となっています．たとえば，68000にはビット操作命令がありますが，他の16ビット・マイコン8086には，こうした命令がまったくなく，68000に比較して大きく見劣りがします．

8086，80186，80286にはビット操作のための命令が，完全に欠落していることは大変に有名（？）なことです．FA（ファクトリ・オートメーション），LA（ラボラトリ・オートメーション）を始めとする制御分野はもちろんのこと，OA（オフィス・オートメーション）関係の分野でも，高度で，高速に走る制御プログラムが必要となってきている現在，このビット操作のための命令がサポートされていないというのは，効率よい制御プログラムを作る上での大きな障

68000と8086の主要3項目の比較

名称	チップ集積度 （トランジスタ数）	内部レジスタ幅	アドレス空間 （直接）
68000	約68000	32ビット	16M
8086	約29000	16ビット	1M

害となります．

これが改善された8086アップワード・コンパチブルなマイコンに μPD70116 (V30) があります．70116には強力なビット操作命令とビット・フィールド操作命令が用意されており，8086の欠点が完全に改善されています．

ビット操作命令は一例ですが，このように強力な命令が68000には多くインプリメントされています．本書は，こうした68000の命令とマシン語，マシン語への変換を詳細に解説しました．

68000のマシン語を1つ1つていねいに解説し，さらにマシン語変換（ハンド・アセンブル）の例題も多く含まれていますから，すべての例題を実際に自分で行なってみることにより，68000のマシン語が完全に理解できるものと思います．

68000のマシン語の理解は，プログラムの開発，デバッグ時に大きな効果をもたらすものと思われます．また，68000のアセンブラなどのシステム・プログラムを作る上でも，マシン語の作り方，生成の仕方を理解することは大きな助けになるものと考えられます．マシン語への変換ができるようになれば，アセンブラがどのような動きをして，コードを生成しているのかを完全に理解できるようになるはずです．

どんな高級言語（日本語プログラミング言語“NIPROL（ニプロール）”；（日本語アセンブラ機能も含む）〔第一システム㈱社製〕などを含めて）を使おうとも，最終的にはマシン語に落とさなければなりません．マシン語は，プログラミング言語の基礎，土台となるものです．

68000のマシン語理解は，プログラムを開発する上で大きな力になるものと確信します．マシン語を理解して，よりよいプログラムが作られるようになることを願って終わりにしたいと思います．

昭和61年１月10日

村山　仁郎

目　次

第１部　68000マシン語プログラミング

第１章　68000の基礎

1．1　68000のレジスタ構成 …… 10
1．2　68000のステータス・レジスタ …… 13
1．3　68000のアドレッシング・モード …… 14

第２章　マシン語変換

2．1　命令の形式 …… 20
2．2　マシン語への変換（マシン語プログラミング） …… 22

第３章　68000の命令セットとマシン語

3．1　データ転送命令 …… 35
3．2　データ転送命令のマシン語プログラミング例 …… 37
3．3　加算，減算命令 …… 40
3.3.1　加算命令のマシン語 …… 40
3.3.2　減算命令のマシン語 …… 45
3．4　加減算命令のマシン語プログラミング例 …… 46
3．5　乗算，除算命令 …… 56

3.5.1　乗算命令のマシン語 …… 56
3.5.2　除算命令のマシン語 …… 58
3.6　乗除算命令のマシン語プログラミング例 …… 60
3.7　比較命令 …… 64
3.7.1　比較命令のマシン語 …… 64
3.8　比較命令のマシン語プログラミング例 …… 68
3.9　クリア命令，テスト命令 …… 73
3.9.1　クリア命令のマシン語 …… 73
3.9.2　テスト命令のマシン語 …… 74
3.10　論理演算命令 …… 75
3.10.1　論理積命令のマシン語 …… 75
3.10.2　論理和命令のマシン語 …… 77
3.10.3　排他的論理和命令のマシン語 …… 79
3.10.4　NOT 命令のマシン語 …… 80
3.11　論理演算命令のマシン語プログラミング例 …… 82
3.12　テスト・アンド・セット命令 …… 88
3.12.1　セマフォーオペレーション …… 88
3.12.2　テスト・アンド・セット（TAS）命令のマシン語 …… 90
3.13　テスト・アンド・セット（TAS）命令のマシン語プログラミング例 …… 91
3.14　BCD 演算命令 …… 95
3.14.1　ABCD 命令のマシン語 …… 95
3.14.2　SBCD 命令のマシン語 …… 97
3.14.3　NBCD 命令のマシン語 …… 98
3.15　BCD 演算命令のマシン語プログラミング例 …… 99
3.16　シフト命令 …… 102
3.16.1　LSL 命令のマシン語 …… 102
3.16.2　LSR 命令のマシン語 …… 104
3.16.3　ASL 命令のマシン語 …… 106
3.16.4　ASR 命令のマシン語 …… 108
3.17　シフト命令のマシン語プログラミング例 …… 110
3.18　回転（ローテート）命令 …… 114
3.18.1　ROL 命令のマシン語 …… 114

3.18.2 ROR 命令のマシン語 ……116
3.18.3 ROXL 命令のマシン語 ……118
3.18.4 ROXR 命令のマシン語 ……120
3.19 回転（ローテート）命令のマシン語プログラミング例 ……122
3.20 ビット操作命令 ……132
3.20.1 BTST 命令のマシン語 ……132
3.20.2 BSET 命令のマシン語 ……134
3.20.3 BCLR 命令のマシン語 ……135
3.20.4 BCHG 命令のマシン語 ……137
3.21 ビット操作命令のマシン語プログラミング例 ……138

第4章 分岐命令

4.1 JMP 命令のマシン語 ……146
4.2 BRA 命令のマシン語 ……148
4.3 Bcc 命令とマシン語 ……149
4.4 DBcc 命令とマシン語 ……153
4.5 分岐命令のマシン語プログラミング例 ……155

第5章 サブルーチンの呼び出し，リターン命令

5.1 JSR 命令とマシン語 ……162
5.2 BSR 命令とマシン語 ……163
5.3 RTS 命令とマシン語 ……164
5.4 RTR 命令とマシン語 ……165
5.5 サブルーチンの呼び出し，リターン命令のマシン語プログラミング例 ……166

第6章　LINK, UNLK 命令

6．1　8086における LINK, UNLK 処理 ……173
6．2　LINK 命令 ……174
6．3　UNLK 命令……175
6．4　LINK, UNLK 命令の使い方……177
6．5　LINK, UNLK 命令のマシン語プログラミング例……179

第7章　トラップ発生命令

7．1　TRAP 命令とマシン語……184
7．2　TRAPV 命令とマシン語 ……188
7．3　CHK 命令とマシン語 ……189

第2部　68000の命令一覧

一覧表……191

はじめに …… 3
参考文献……268
索　引……269

第1部
68000マシン語プログラミング

第1章

68000の基礎

1.1 68000のレジスタ構成

まず最初に，68000の基礎事項として，レジスタ構成がどうなっているのかを見てみましょう．わかりやすくするために，例題形式で話を進めます．

例題1 68000のレジスタ構成を示し，これを説明しなさい．

解き方

図1.1に68000のレジスタ構成を示します．データ・レジスタがD0からD7までの8個，アドレス・レジスタがA0からA6までの7個，そして2個のスタック・ポインタ，1個のプログラム・カウンタ，1個のステータス・レジスタが用意されています．

データ・レジスタ（D0～D7）は32ビット長のレジスタで，データ処理用の汎用データ・レジスタです．処理可能な単位は，バイト，ワード，ロング・ワードで，バイト・レジスタとしても，ワード・レジスタとしても，ロング・ワード・レジスタとしても動作することが可能です．

バイト・レジスタとして動作する場合は32ビット中の下位8ビットが用いられ，ワード・レジスタとして動作する場合は，下位16ビットが用いられます．

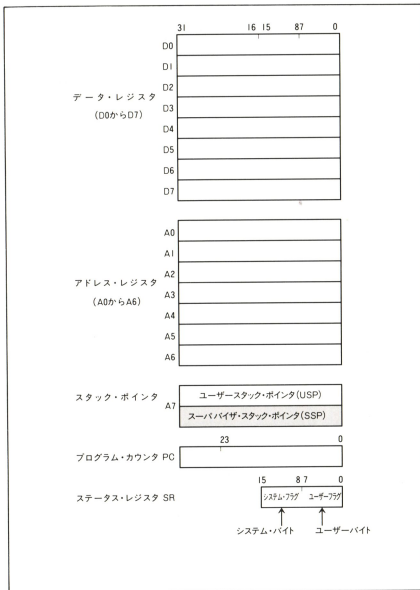



図1.1　68000のレジスタ構成

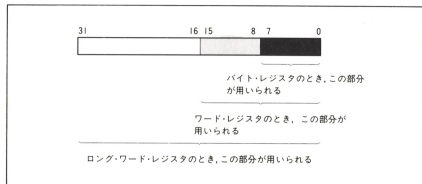



図1．2　バイト・レジスタ，ワード・レジスタ，ロング・ワード・レジスタで用いられる部分

残りの上位ビットは変化しません．これを**図1．2**に示します．

アドレス・レジスタ（A0～A6）は32ビット長のレジスタで，アドレス・オペランドを指定するのに用いられ，バイト・サイズで使用することはできません．

スタック・ポインタは32ビット長のレジスタで，スーパバイザ用とユーザー用に2個用意されており，ステータス・レジスタSR中のSビットで，どちらか一方の状態が示されます．S＝0のときユーザー状態で，スタック・ポインタとしてはユーザースタック・ポインタ（USP）が用いられ，S＝1のときはスーパバイザ状態で，SPとしてはスーパバイザ・スタック・ポインタ（SSP）が用いられます．

プログラム・カウンタPCは32ビット長ですが，実際には，下位24ビットを外部で用いることが可能となっています．したがって，68000のアドレス可能な範囲は16Mまでで，$000000から$FFFFFF番地までとなります．

68000の命令サイズは2バイト命令，4バイト命令，6バイト命令，8バイト命令，10バイト命令とあり，最小の命令サイズは2バイトからなります．

これらの命令は偶数番地のアドレスに格納する必要があり，プログラム・カウンタにも偶数アドレスの格納が必要で，奇数アドレスの場合，アドレス・エラーが発生します（この点が8086と異なっており，8086では命令サイズは1バイトから6バイトまであり，メモリ中のどの番地に置くこともできました．すなわち，奇数アドレスから命令を格納しても問題は生じませんでした．ただ，バス・サイクルが余分に必要とはなります）．

ステータス・レジスタは16ビットから構成され，システム・フラグが入るシステム・バイト（上位バイト）とユーザーフラグが入るユーザーバイト（下位バイト）からなっています．

1.2　68000のステータス・レジスタ

次に68000のステータス・レジスタSRを見てみましょう．

例題2　ステータス・レジスタのフォーマットを示しなさい．

解き方

図1.3にステータス・レジスタのフォーマットを示します．ユーザーバイトはおのおの，

C………キャリーフラグ
V………オーバーフローフラグ
Z………ゼロ・フラグ
N………ネガティブ・フラグ
X………エクステンド・フラグ

を意味します．

また，システム・バイトは次のようになっています．

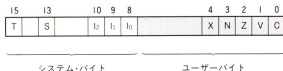



図1.3　ステータス・レジスタのフォーマット

$I_0 \sim I_2$…………割込みマスク
S………………スーパバイザ状態フラグ
T………………トレース・モード・フラグ

割込みマスクは，ここで指定されたレベルより優先度の高い割込み要求を受け付け，同じレベルか低い優先度の割込み要求は受け付けなくするためのマスク用ビットです．

スーパバイザ状態フラグSは，S＝1でスーパバイザ状態，S＝0でユーザー状態を示します．Sの値によって，スタック・ポインタはUSPまたはSSPが選択されて用いられるのは前述したとおりです．

トレース・モード・フラグTは，シングル・ステップ，トレースを行なうかどうかの指定ビットで，T＝1にセットすると，命令実行のたびに例外状態に入り（スーパバイザ状態），例外ベクタ・アドレスをもとにしてシングル・ステップ処理ルーチン，トレース処理ルーチンに制御が渡されます．

1.3　68000のアドレッシング・モード

ここでは，68000のアドレッシング・モードについて考えて見ましょう．

例題3　68000のアドレッシング・モードについて説明しなさい．

▶ 解き方 ◀

大きく6とおりのアドレッシング・モードに分類でき，それらに複数の形式が対応し，トータルで14種類のアドレッシング・モードがあります（**表1.1**）．

(1)　**データ・レジスタ直接**

データ・レジスタDnを直接オペランドに用いるモードで，

MOVE. W　D1, D2

はデータ・レジスタD1の内容をD2へ転送します．実効アドレスEA（21ページ参照）は，

EA＝Dn

表1.1 68000アドレッシング・モード

(1)	データ・レジスタ直接	レジスタ直接
(2)	アドレス・レジスタ直接	
(3)	アドレス・レジスタ間接	レジスタ間接
(4)	ポスト・インクリメント・アドレス・レジスタ間接	
(5)	プリ・デクリメント・アドレス・レジスタ間接	
(6)	ディスプレースメント付アドレス・レジスタ間接	
(7)	インデックス・ディスプレースメント付 アドレス・レジスタ間接	
(8)	アブソリュート・ショート	アブソリュート
(9)	アブソリュート・ロング	
(10)	ディスプレースメント付プログラム・カウンタ相対	プログラム・カウンタ相対
(11)	インデックス・ディスプレースメント付 プログラム・カウンタ相対	
(12)	即値	即値
(13)	クイック即値	
(14)	インプライド	インプライド

となります。

(2) アドレス・レジスタ直接

アドレス・レジスタAnが直接オペランドとなり，実効アドレスEAは，

EA＝An

となります。

たとえば，

MOVE. W　A1, D1

はアドレス・レジスタA1の内容をD1に転送します．

(3) アドレス・レジスタ間接

アドレス・レジスタAnがポイントするメモリがアドレッシング，アクセスされます．実効アドレスEAは，

EA＝(An)

となります．

たとえば，

MOVE. W　(A1), D1

はアドレス・レジスタA1がポイントするメモリ内容がD1に転送されます．A1に1000H*が格納されていれば，1000H番地のメモリのワード・データがD1に転送

＊HはHexadecimal(16進数)の意味．ニーモニック中の＄も同様．

されます．

(4) ポスト・インクリメント・アドレス・レジスタ間接

これはアドレス・レジスタ間接(3)の動作後に，アドレス・レジスタがインクリメントされます．インクリメントの幅はバイト・オペレーションの場合＋1，ワード・オペレーションの場合＋2，ロング・ワード・オペレーションの場合＋4となります．実効アドレスEAは，

EA＝(An)，An←An＋N（N＝1または2または4）

たとえば，

MOVE. W (A1)＋, D1

はA1がポイントするメモリ内容をD1に転送し，A1の内容が＋2だけインクリメントされます．A1に1000Hが格納されていれば，1000H番地のメモリのワード・データをD1に転送し，A1は1002Hに＋2だけインクリメントされます．

(5) プリ・デクリメント・アドレス・レジスタ間接

アドレス・レジスタAnを前もってデクリメント（いいかえればプリ・デクリメント）し，そのAnがポイントするメモリがアドレッシング，アクセスされます．実効アドレスEAは，

An←An－N（N＝1または2または4），EA＝(An)

となります．

デクリメントの幅Nは，バイト・オペレーションのときN＝1，ワード・オペレーションのときN＝2，ロング・ワード・オペレーションのときN＝4となるのは，ポスト・インクリメントの場合と同様です．

たとえば，

MOVE. W －(A1), D1

はA1の内容を－2デクリメントしてから，そのA1がポイントするメモリ内容をワードでD1レジスタに転送します．A1に1000Hが格納されていれば，－2デクリメントされて，0FFEHにA1の値はなり，この0FFEH番地のワード・データがD1に転送されます．

(6) ディスプレースメント付アドレス・レジスタ間接

16ビットのディスプレースメントとアドレス・レジスタAnの内容との和が実効アドレスとなります（この際ディスプレースメントDISP16は32ビットに符号拡張されます）．

EA＝(An)＋DISP16

たとえば，

MOVE. W　$100(A1), D1

はDISP16が＄100ですから，A1の値に100Hを加算して，そのA1がポイントするメモリのワード・データをD1に転送します．

A1に1000Hが格納されていれば，(A1)＋DISP16＝1000H＋100H＝1100H，1100H番地のメモリのワード・データをアクセスし，これをD1に転送します．

(7)　インデックス・ディスプレースメント付アドレス・レジスタ間接

このアドレッシング・モードは実効アドレスEAの計算時に，インデックス・レジスタの値，アドレス・レジスタの値，ディスプレースメントの値をすべて含めて加算します．ディスプレースメントは8ビットのDISP8ですが，加算時に32ビットに符号拡張されます．インデックス・レジスタには，データ・レジスタもアドレス・レジスタも用いることができます．

実効アドレスEAは，

EA＝(An)＋(Xn)＋DISP8

ここでXnはインデックス・レジスタを指します．

たとえば，

MOVE. W　$10(A1, D0), D1

はDISP8が＄10ですから，A1の値に10Hを加算して，さらにインデックス・レジスタD0の値を加算して，実効アドレスEA値を求め，これがポイントするメモリのワード・データをD1に転送します．

A1に1000H，D0に2000Hが格納されていれば，

EA ＝(An)＋(Xn)＋DISP8
＝(A1)＋(D0)＋DISP8
＝1000H＋2000H＋10H
＝3010H

となり，3010H番地のメモリのワード・データがD1に転送されます．

(8)　アブソリュート・ショート

次の1ワード，すなわち，実効アドレス拡張ワードの1ワードが実効アドレスとなります．

EA＝次の1ワード

たとえば，

MOVE. W　$100, D1

は絶対番地100Hの内容をワードでD1に転送します．

(9)　アブソリュート・ロング

次の2ワード，すなわち実効アドレス拡張ワードの2ワードが実効アドレスとなり，

EA＝次の2ワード

となります．

たとえば，

MOVE. W　$012000，D1

は，絶対番地12000Hの内容をワードでD1に転送し，12000H番地のメモリ・アクセスにはアブソリュート・ロング，すなわち，命令の実効アドレス拡張ワードの2ワードにわたって格納されている12000Hの値を用いて，アドレッシングが行なわれます．

(10)　ディスプレースメント付プログラム・カウンタ相対

プログラム・カウンタPCの内容とディスプレースメントDISP16との和が実効アドレスとなります．

EA＝(PC)＋DISP16

ディスプレースメントは，16ビットのDISP16ですから，カレントPCから±32Kの範囲にアクセスが許されることになります(内部ではEA計算時にDISP16は32ビットに符号拡張される)．PC内容は拡張ワードアドレスを指しています．

たとえば，PC値が1000Hで，ディスプレースメント値が100Hであれば，1100H番が実効アドレスとなります．

(11)　インデックス・ディスプレースメント付プログラム・カウンタ相対

実効アドレス計算で，上記の(10)にインデックス・レジスタ（Xn）の値がさらに加わった場合のアドレッシング・モードです．この場合，ディスプレースメントは8ビットのDISP8となります．

実効アドレスEAは，

EA＝(PC)＋(Xn)＋DISP8

となります．

たとえば，PC値が1000Hで，ディスプレースメント値が10Hで，A0の値が2000Hであれば，3010H番地が実効アドレスとなります．

(12)　即　値

1ワード，または2ワードの実効アドレス拡張ワードに即値が入り，

即値＝次の1ワードまたは次の2ワード

となります．

たとえば，

MOVE. W　#$1000，D1

は即値＄1000がD1に転送（ワードで）されます．

(13)　クイック即値

クイック（quick）即値アドレッシング・モードは，即値が最初のオペレーション・ワード中にあって，拡張ワード中にはないので，オペランドのリード・サイクル分だけ実行をクイックに行なうことができます．しかし，扱うことができる即値は±128，または1～8と限定されます．

これが使用できる命令は，

ADDQ（add quick）

MOVEQ（move quick）

SUBQ（subtract quick）

の3命令となります．

たとえば，

MOVEQ　$10，D1

は即値＄10がD1に転送されます．

(14)　インプライド

暗黙にオペランドがオペコードによって決まってしまうモードをいいます．たとえば**RTS**命令ではサブルーチンからのリターンですから，暗黙的にSPが用いられます．

以上が68000の14とおりのアドレッシング・モードです．

第2章

マシン語変換

2.1 命令の形式

68000の命令形式は**図2.1**に示すように,

① オペレーション・ワード
② 即値オペランド
③ ソース実効アドレス・オペランド
④ デスティネーション実効アドレス・オペランド

の4種類の組み合わせからなり,命令の長さは1ワードから5ワードまでのいずれかになります(即値オペランド・ワードとソース実効アドレス・オペランド・ワードは,同時には存在しません).

(1) オペレーション・ワード

オペレーション・ワードは1ワードで,どのオペレーションを行なうかのオペレーション・コード(オペコード:op-code)と実効アドレスを指定するフィ

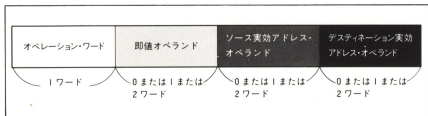



図2.1 命令の形式,フォーマット

ールドの2つのフィールドから構成されています．

(2)　即値オペランド・フィールド

即値オペランド・フィールドは，即値が入るフィールドで，即値オペレーションのときにこのフィールドが用いられ，1ワードまたは2ワード長の即値データが入り，即値オペレーションでない場合は，このフィールドは必要ありませんから存在しません．

(3)　ソース実効アドレス・オペランドとデスティネーション実効アドレス・オペランド・フィールド

ソース実効アドレス・オペランド・フィールド，デスティネーション実効アドレス・オペランド・フィールドは，おのおのソースとデスティネーションの実効アドレス・オペランドを必要とする命令で用いられるフィールドで，1ワードまたは2ワード長で，必要ないときはもちろん存在しません．

以上の4種類のフィールドの組み合わせからマシン語が作られていきます．マシン語の最初の1ワードは，オペレーション・ワードで，この16ビット中に6ビットからなる実効アドレス・フィールドがあり，たいていの場合，このフィールドによって実効アドレスが指定されます．この実効アドレス・フィールドは，**図2.2**に示すように，アドレッシング・モード・フィールド（3ビット長）とレジスタ・フィールド（3ビット長）とからなっており，おのおのアドレッシング・モードと使用レジスタを指定します．

メモリ・アドレッシング・モードは，アドレッシング・モード・フィールドとレジスタ・フィールドのビット・パターンによって決定されます．これを表

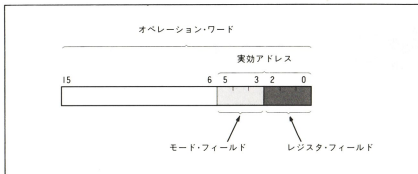



図2.2　一般的なオペレーション・ワード中の実効アドレス・フィールド

表2.1　モード，レジスタ・フィールドによって決められるアドレッシング・モード

モード・フィールド	レジスタ・フィールド	アドレッシング・モード
0 0 0	Dn	データ・レジスタ直接
0 0 1	An	アドレス・レジスタ直接
0 1 0	An	アドレス・レジスタ間接
0 1 1	An	ポスト・インクリメント・アドレス・レジスタ間接
1 0 0	An	プリ・デクリメント・アドレス・レジスタ間接
1 0 1	An	ディスプレースメント付アドレス・レジスタ間接
1 1 0	An	インデックス，ディスプレースメント付アドレス・レジスタ間接
1 1 1	0 0 0	アブソリュート・ショート
1 1 1	0 0 1	アブソリュート・ロング
1 1 1	0 1 0	ディスプレースメント付プログラム・カウンタ相対
1 1 1	0 1 1	インデックス，ディスプレースメント付プログラム・カウンタ相対
1 1 1	1 0 0	即　値

2.1に示します．この表は非常に重要なもので，マシン語をつくるときの鍵となりますので，しっかりと理解してください．

68000のアドレッシング・モードはトータルで14種類のモードがあり，表2.1では，このうち，

クイック即値

インプライド

の２つのモードがありません．これはクイック即値モードでは，オペレーション・ワード中に即値データが存在し，インプライド・モードでは，オペコードによって暗黙的にオペランドが指定されますから，これらの２つのモードは表2.1には含まれていないわけです．

2.2　マシン語への変換（マシン語プログラミング）

次に，最も簡単な例として，ソースからデスティネーションへデータを転送する**MOVE**命令のマシン語を考えてみましょう．

例題4

次の命令をマシン語にハンドアセンブルしなさい．

```
MOVE.W  D1, D2
```

解き方

MOVE命令のマシン語フォーマットは，

で，ソース・オペランドの指定は，ビット0から5までで行ない，デスティネーション・オペランドの指定は，ビット6から11までの6ビットで行ない，アドレッシング・モードは，表2.1で示したようにモード，レジスタ・フィールドのビットを選択すれば指定されます．サイズ・フィールドは，転送するオペランドのサイズ，大きさを指定するもので，

0 1……バイト・オペレーション

1 0……ロング・ワード・オペレーション

1 1……ワード・オペレーション

を意味します．

MOVE. W　D1，D2では，第1オペランドのD1がソース・オペランドで，第2オペランドのD2がデスティネーション・オペランドとなり，D1の内容をD2へワード転送し，アドレッシング・モードは，データ・レジスタ直接となります．

ソース・フィールドにはD1を表わすモード・フィールド＝▼000▼，レジスタ・フィールド＝▼001▼を，デスティネーション・フィールドには，D2を表わすモード・フィールド＝▼000▼，レジスタ・フィールド＝▼010▼をマシン語フォーマットに代入します．

また，サイズ・フィールドには，ワード転送ですからワード・オペレーションの▼11▼を代入してやります．以上をまとめると，

(注)HはHexの意味．以下同じ．

3401Hが**MOVE. W　D1，D2**のマシン語となります．

例題5 次のプログラムをマシン語に変換しなさい。

```
ORG        $3000
MOVE.W     A1, D1
MOVE.W     (A1), D1
MOVE.W     (A1)+, D1
MOVE.W     -(A1), D1
MOVE.W     $100(A1), D1
MOVE.W     $10(A1, D0), D1
MOVE.W     $100, D1
MOVE.W     $012000, D1
MOVE.W     #$1000, D1
MOVEQ      #$10, D1
END
```

解き方

MOVEQ以外の**MOVE**命令のマシン語フォーマットは，前述したように，

で，これを用いてマシン語におのおの変換していくことになります．

MOVE. W A1, D1は，サイズ・フィールドはワード・オペレーションでサイズ＝▼11▼，ソース・フィールドはモード＝アドレス・レジスタ直接＝▼001▼，レジスタ＝A1＝▼001▼，デスティネーション・フィールドは，モード＝データ・レジスタ直接＝▼000▼，レジスタ＝D1＝▼001▼となり，以上をマシン語フォーマットに代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	1	0	0	0	0	0	1	0	0	1	………3209H

3209HがMOVE. W　A1, D1のマシン語となります．

MOVE. W　(A1), D1命令は，サイズはワード操作でサイズ＝▼11▼，ソース・フィールドはモード＝アドレス・レジスタ間接＝▼010▼，レジスタ＝A1＝▼001▼，デスティネーション・フィールドはモード＝データ・レジスタ直接＝▼000▼，レジスタ＝D1＝▼001▼となり，以上をマシン語フォーマットに代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	1	………3211H

3211HがMOVE.W　(A1), D1のマシン語となります．

MOVE.W　(A1)＋, D1命令はサイズ＝ワード・オペレーション＝▼11▼，ソース・フィールドはモード＝ポスト・インクリメント・アドレス・レジスタ間接＝▼011▼，レジスタ＝A1＝▼001▼，デスティネーション・フィールドは前の命令と同様で，これらをマシン語フォーマットに代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	1	0	0	0	0	1	1	0	0	1	………3219H

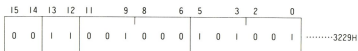
3219HがMOVE. W　(A1)＋, D1のマシン語となります．

MOVE. W　－(A1), D1命令は，サイズ＝ワード・オペレーション＝▼11▼，ソース・フィールドはモード＝プリ・デクリメント・アドレス・レジスタ間接＝▼100▼，レジスタ＝A1＝▼001▼，デスティネーション・フィールドは前の命令と同様で，これらをマシン語フォーマットに代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	1	………3221H

3221HがMOVE. W　－(A1), D1のマシン語となります．

MOVE. W　$100(A1), D1命令はサイズ＝ワード・オペレーション＝▼11▼，ソース・フィールドはモード＝ディスプレースメント付アドレス・レジスタ間接＝▼101▼，レジスタ＝A1＝▼001▼，デスティネーション・フィールドは前の命令と同様で，これらをマシン語フォーマットに代入して，

15 14	13 12	11 　9 8 　6	5 　3 2 　0	
0 0	1 1	0 0 1 0 0 0	1 0 1 0 0 1	………3229H

3229Hが**MOVE.W　$100(A1)，D1**のオペレーション・ワードとなります．しかし，これだけではマシン語は完全ではありません．ディスプレースメント値$100をソース実効アドレス・オペランド・フィールドにセットしてやる必要があります．これはワードで**0100H**となりますから，以上をまとめて，

32290100H

が**MOVE. W　$100(A1)，D1**のマシン語となります．

MOVE.W　$10(A1，D0)，D1命令はインデックス，ディスプレースメント付アドレス・レジスタ間接アドレッシング・モードで，この場合インデックス・レジスタを指定するため，オペレーション・ワードのほかに，１ワードの実効アドレス・ワードが必要となり，このフォーマットを**図2.3**に示します．

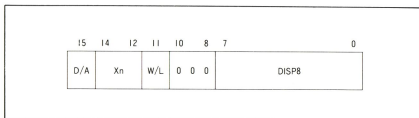



図２．３　インデックス・レジスタ指定ワード・フォーマット

図2.3において，

D/A ………０でデータ・レジスタ
　　　　　　１でアドレス・レジスタ

Xn …………インデックス・レジスタ番号

W/L ………０でインデックス・レジスタのワード(下位)が符号拡張
　　　　　　１でインデックス・レジスタのロング・ワードがそのまま用いられる

DISP8………８ビット長のディスプレースメント
　　　　　　(DISPLACEMENT)

を意味します．

オペレーション・ワード・フォーマット（MOVE命令）；

15	14	13 12	11　　9 8　　6	5　　3 2　　0
0	0	サイズ	デスティネーション （レジスタ）　（モード）	ソース （モード）　（レジスタ）

において，サイズは**MOVE.W**ですからワード・オペレーションとなり，サイズ・フィールド＝▼11▼，デスティネーション・フィールドはレジスタ＝D1＝▼001▼，モード＝データ・レジスタ直接＝▼000▼となります．

一方，ソース・フィールドはモード＝インデックス，ディスプレースメント付アドレス・レジスタ間接＝▼110▼，レジスタ＝A1＝▼001▼となり，これらを代入してオペレーション・ワードは，

3231Hとなります．

次に，インデックス・レジスタ指定拡張ワードが続き，図2.3のフォーマットにおいて，D／Aはインデックス・レジスタD0だからデータ・レジスタで0，Xn＝D0＝▼000▼，W／L＝▼0▼，DISP8＝$10を代入して，

0010Hがオペレーション・ワードに続くインデックス・レジスタ指定拡張ワードのマシン語となります．以上をまとめて**MOVE.W　$10(A1，D0)，D1**のマシン語は，

32310010H

となります．

MOVE. W　$100, D1命令はアブソリュート・ショート・アドレッシング・モードで，オペレーション・ワードとそれに続く位置に，ワードで実効アドレスEAが格納されます（すなわちEA＝次の１ワード）．

オペレーション・ワードは，サイズ・フィールド＝ワード＝▼11▼，デスティネーション・フィールドは，レジスタ＝D1＝▼001▼，モード＝データ・レジスタ直接＝▼000▼，ソース・フィールドはアブソリュート・ショートでモード＝▼111▼，レジスタ＝▼000▼となり，これらを代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	1	0	0	0	1	1	1	0	0	0	……3238H

3238Hがマシン語の第１ワード目の値となります．次のワードに$100を持ってくればよく，**MOVE. W　$100, D1**のマシン語は，

32380100H

となります．

MOVE. W　$012000, D1命令はアブソリュート・ロング・アドレッシング・モードで，オペレーション・ワードとそれに続く位置に，ロング・ワードで実行アドレスEAが格納されます（すなわちEA＝次の２ワード）．

オペレーション・ワードは，サイズ・フィールド＝ワード＝▼11▼，デスティネーション・フィールドは，レジスタ＝D1＝▼001▼，モード＝データ・レジスタ直接＝▼000▼，ソース・フィールドは，アブソリュート・ロングでモード＝▼111▼，レジスタ＝▼001▼となり，これらを代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	1	0	0	0	1	1	1	0	0	1	……3239H

3239Hがマシン語の第１ワード目の値となります．次の２ワードに$012000のロング・ワードを持ってくればよく，**MOVE. W　$012000, D1**のマシン語は，

323900012000H

となることがわかります．

MOVE.W　#$1000, D1命令は即値アドレッシング・モードで，オペレーショ

ン・ワードの次に即値オペランド・フィールドがあり，ここに即値データが入ります．

オペレーション・ワードは，サイズ・フィールド＝ワード・オペレーション＝▼11▼，デスティネーション・フィールドはレジスタ＝D1＝▼001▼，モード＝データ・レジスタ直接＝▼000▼，ソース・フィールドは即値アドレッシング・モードですから，モード＝▼111▼，レジスタ＝▼100▼となり，これらを代入してオペレーション・ワードは，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	1	0	0	0	1	1	1	1	0	0	………323CH

323CHがマシン語（オペレーション・ワード）となります．次の即値オペランドにワードで$1000が格納されますから，**MOVE.W #$1000, D1**命令のマシン語は，

323C1000H

となります．

MOVEQ #$10, D1命令はクイック転送命令で，クイック即値アドレッシング・モードでは，即値データはオペレーション・ワードそれ自体の中に入っており，即値オペランド拡張ワードは使われません．

MOVEQのマシン語フォーマットは，

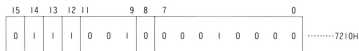
15	14	13	12	11　　9	8	7　　　　　　　　　0
0	1	1	1	レジスタ	0	即値データ

で，

レジスタ・フィールド………データ・レジスタDnのnの値

即値データ・フィールド……8ビット長の即値

を意味し，8ビットの即値データが32ビットのロング・ワードに符号拡張され，ロング・ワードがDnレジスタに転送されます．**MOVEQ #$10, D1**はレジスタ・フィールド＝D1＝▼001▼，即値データ・フィールド＝$10ですから，これらをマシン語フォーマットに代入して，

15	14	13	12	11		9	8	7							0	
0	1	1	1	0	0	1	0	0	0	0	1	0	0	0	0	………7210H

7210HがMOVEQ　#$10，D1のマシン語となります．

以上をまとめると，答は**表2.2**のようになります．

表2.2　例題5のハンド・アセンブル・リスト

	00003000	ORG	$3000
003000	3209	MOVE. W	A1, D1
003002	3211	MOVE. W	(A1), D1
003004	3219	MOVE. W	(A1)+, D1
003006	3221	MOVE. W	−(A1), D1
003008	32290100	MOVE. W	$100(A1), D1
00300C	32310010	MOVE. W	$10(A1, D0), D1
003010	32380100	MOVE. W	$100, D1
003014	323900012000	MOVE. W	$012000, D1
00301A	323C1000	MOVE. W	#$1000, D1
00301E	7210	MOVEQ	#$10, D1
		END	

第3章

68000の命令セットとマシン語

68000の命令セットを大別すると,

① データ転送命令
② 算術演算命令
③ 論理演算命令
④ ビット操作命令
⑤ プログラム制御命令
⑥ CPU制御命令

のようになりますが,これらの命令の中身を他のマイコン8086と比較して,68000の命令の特徴をみることにしましょう.

例題6

68000の命令セットを8086の命令セットと比較,検討しなさい.

解き方

68000と8086の命令を,データ転送命令,算術演算命令,論理演算命令,ビット操作命令,ストリング操作命令,プログラム制御命令,CPU制御命令に分類し,表にしたものが**表3.1**です.

表3.1　68000と8086の命令比較

項　目	68000	8086
データ転送命令	**MOVE**(データ転送) **MOVEA**(アドレス・レジスタに転送) **MOVEP**(ペリフェラル・データ転送) **MOVEQ**(即値クイック転送) **MOVEM**(複数レジスタとの転送) **EXG**(レジスタ内容の交換) **LEA**(実効アドレスのロード) **SWAP**(上位ワードと下位ワードの入れ換え) **PEA**(実効アドレスのプッシュ) **LINK**(リンク，スタックに領域確保) **UNLK**(アンリンク，スタックの領域解放)	**MOV**(データ転送，即値転送も含む) **PUSH**(プッシュ) **POP**(ポップ) **XCHG**(交換) **IN**(入力) **OUT**(出力) **XLAT**(置換) **LEA**(実効アドレスのロード) **LDS**(DSとレジスタへロード) **LES**(ESとレジスタへロード) **LAHF**(フラグをAHへロード) **SAHF**(AHをフラグ・レジスタへ転送) **PUSHF**(フラグ・プッシュ) **POPF**(フラグ・ポップ)
算術演算命令	**ADD**(加算) **ADDA**(アドレス・レジスタとの加算) **ADDI**(即値加算) **ADDQ**(即値クイック加算) **ADDX**(拡張フラグ付加算) **SUB**(減算) **SUBA**(アドレス・レジスタからの減算) **SUBI**(即値減算) **SUBQ**(即値クイック減算) **SUBX**(拡張フラグ付減算) **MULS**(符号付き乗算) **MULU**(符号なし乗算) **DIVS**(符号付き除算) **DIVU**(符号なし除算) **CLR**(オペランド内容をゼロ・クリア) **NEG**(負数化) **NEGX**(拡張フラグX付負数化) **TAS**(テスト・アンド・セット) **TST**(オペランド内容を0とテスト) **EXT**(符号拡張) **CMP**(比較)	**ADD**(加算) **ADC**(キャリー付加算) **INC**(インクリメント) **AAA**(加算アスキー補正) **DAA**(加算10進補正) **SUB**(減算) **SBB**(ボロー付減算) **DEC**(デクリメント) **NEG**(負数化) **CMP**(比較) **AAS**(減算アスキー補正) **DAS**(減算10進補正) **MUL**(符号なし乗算) **IMUL**(符号付き乗算) **AAM**(乗算アスキー補正) **DIV**(符号なし除算) **IDIV**(符号付き除算) **AAD**(除算アスキー補正) **CBW**(バイトをワードへ拡張) **CWD**(ワードをダブル・ワードへ拡張)

算術演算命令	CMPA(アドレス・レジスタとの比較) CMPM(メモリ内容の比較) CMPI(即値との比較) ABCD(BCD加算) SBCD(BCD減算) NBCD(BCD負数化)	
論理演算命令	AND(論理積) ANDI(即値と論理積) OR(論理和) ORI(即値と論理和) EOR(排他的論理和) EORI(即値と排他的論理和) NOT(1の補数化，ビットをすべて反転) ASL(左へ算術シフト) ASR(右へ算術シフト) LSL(左へ論理シフト) LSR(右へ論理シフト) ROL(左へ回転) ROR(右へ回転) ROXL(拡張フラグXも含めて左へ回転) ROXR(拡張フラグXも含めて右へ回転)	NOT(1の補数化，ビットをすべて反転) SHL(左へ論理シフト) SAL(左へ算術シフト) SHR(右へ論理シフト) SAR(右へ算術シフト) ROL(左へ回転) ROR(右へ回転) RCL(キャリーも含めて左へ回転) RCR(キャリーも含めて右へ回転) AND(論理積) TEST(テスト．ANDをとってフラグのみ変化) OR(論理和) XOR(排他的論理和)
ビット操作命令	BTST(ビットのテスト) BSET(ビットのテスト後，1にセット) BCLR(ビットのテスト後，ゼロ・クリア) BCHG(ビットのテスト後，反転)	
ストリング操作命令		REP(リピート・プリフィックス) MOVS(ストリング転送) CMPS(ストリング比較) SCAS(ストリング・スキャン) LODS(ストリング・ロード) STOS(ストリング・ストア)
プログラム制御命令	Bcc(条件付ブランチ) BCC, BCS, BEQ, BGE, BGT, BHI, BLE, BLS, BLT, BMI,	CALL(プロシージャのコール) JMP(無条件ジャンプ) 条件付きジャンプ

項　目	68000	8086
プログラム制御命令	BNE, BPL, BVC, BVS DBcc(条件テスト，デクリメント，ブランチ) DBCC, DBCS, DBEQ, DBF, DBGE, DBGT, DBHI, DBLE, DBLS, DBLT, DBMI, DBNE, DBPL, DBRA, DBT, DBVC, DBVS Scc(条件セット) SCC, SCS, SEQ, SF, SGE, SGT, SHI, SLE, SLS, SLT, SMI, SNE, SPL, ST, SVC, SVS BRA(ブランチ) BSR(サブルーチンへブランチ) JMP(ジャンプ) JSR(サブルーチンへジャンプ) RTR(リターン，コンディション・コード復元) RTS(サブルーチンからリターン) RTE(例外処理からリターン) TRAP(トラップ発生) TRAPV(オーバーフローでトラップ発生) CHK(境介チェック)	JE, JZ, JL, JNGE, JLE, JNG, JB, JNAE, JBE, JNA, JP, JPE, JO, JS, JNE, JNZ, JNL, JGE, JNLE, JG, JNB, JAE, JNBE, JA, JNP, JPO, JNO, JNS RET(リターン) LOOP(ループ) LOOPZ(ゼロの間ループ) LOOPE(等しい間ループ) LOOPNZ(≠0の間ループ) LOOPNE(等しくない間ループ) JCXZ(CXがゼロでジャンプ) INT(割込み発生) INTO(オーバーフローで割込み発生) IRET(割込みリターン)
CPU制御命令	RESET(リセット) STOP(SRへロードしてストップ) MOVE to/from USP(USPとの転送) MOVE to SR(SRへ転送) MOVE from SR(SRから転送) MOVE to CCR(CCRへ転送) ANDI to SR(即値をSRに論理積) ANDI to CCR(即値をCCRに論理積) ORI to SR(即値をSRに論理和) ORI to CCR(即値をCCRに論理和) EORI to SR(即値をSRに排他的論理和) EORI to CCR(即値をCCRに排他的論理和) NOP(何もしないで次の命令へ行く)	CLC(キャリークリア) STC(キャリーセット) CMC(キャリー反転) CLD(DFフラグ・クリア) STD(DFフラグ・セット) CLI(IFフラグ・クリア) STI(IFフラグ・セット) HLT(ホールト) WAIT(ウェート) LOCK(バス・ロック) ESC(エスケープ) NOP(何もしないで次の命令へ行く)

データ転送命令での大きな特徴は，68000では複数レジスタとの転送命令である**MOVEM**や，スタック上の領域確保や解放のための**LINK**，**UNLK**命令があるのに対し，8086では**I／O**アドレス空間での入出力命令である**IN**命令，**OUT**命令，セグメント・レジスタへの転送命令（**MOV**命令，**LDS**命令，**LES**命令）などがある点です．

算術演算命令では，どちらも大体同じ種類の命令が用意されていますが，オペレーションのサイズは，68000ではバイト，ワード，ロング・ワード（32ビット）なのに対し，8086ではバイトかワードのオペレーションのみで，32ビットのロング・ワード・オペレーションは不可能です．

68000の**TAS**（テスト・アンド・セット）命令は，マルチ・プロセッサ用の命令で，セマフォーオペレーションで使用します．8086にはテスト・アンド・セット命令はありませんが，**LOCK**命令と**XCHG**命令を一緒に使用してセマフォーオペレーションを実現します．論理演算命令は，どちらもほぼ同じ命令がサポートされています．

ビット操作命令では，68000がこれをサポートしているのに対し，8086ではいっさいビット操作命令は存在しません．また，ストリング操作命令では逆に，8086がこれをサポートしているのに対し，68000ではこの種の命令はサポートされていません．

プログラム制御命令では，68000では条件付き命令が**Bcc**，**DBcc**，**Scc**と多く用意されているのに対し，8086では条件付きジャンプ命令のみです．また，境介チェックに用いる68000の**CHK**命令に相当するものは，8086には存在しません．

CPU制御命令では，68000には**RESET**，**STOP**など特徴ある命令が用意されています．

3.1　データ転送命令

データ転送命令には，**MOVE**，**MOVEA**，**MOVEM**，**MOVEP**，**MOVEQ**，**EXG**，**SWAP**，**LINK**，**UNLK**，**LEA**，**PEA**命令があります．

(1)　MOVE命令

MOVE命令はソース・オペランドの内容をデスティネーションへ転送する命令で，オペランドのサイズ，すなわち**MOVE**命令のオペレーションのサイズはB（バイト），W（ワード），L（ロング・ワード）があります．

(2) MOVEA命令

MOVEA命令は，デスティネーションにアドレス・レジスタAnを指定する場合に用いられ，ソース内容がAnに転送されます．サイズはW，Lが許されます．

(3) MOVEM命令

MOVEM (MOVE Multiple Registers) 命令は，複数の内部レジスタ群とメモリ間でブロック転送するもので，サイズはW，Lを指定することができます．

(4) MOVEP命令

MOVEP (MOVE Peripheral) 命令は，8ビット用に作られたプログラマブル周辺LSIとのデータ転送用に便利な命令で，これを使って68000の16ビット・バスにインターフェイスされた8ビット用周辺LSIとデータのやり取りを行なうことができます．サイズはW，Lを指定することができます．

(5) MOVEQ命令

MOVEQ (MOVE Quick) 命令は，オペレーション・ワード中の即値（8ビット）をデータ・レジスタDnに転送する命令で，8ビットの即値はロング・ワードに符号拡張され，転送されます．即値のオペランド・サイズは当然バイトで，それ以外は許されません．

(6) EXG命令

EXG (EXchanGe Registers) 命令は，2つのレジスタ間でデータ交換を行ない，サイズはロング・ワードLのみ許され，データ交換はすべて32ビットのロング・ワードで行なわれます．

(7) SWAP命令

SWAP命令は同一のデータ・レジスタ内の上位ワードと下位ワードのデータの入れ換え，すなわちスワップをします．

(8) LINK，UNLK命令

LINK (LINKとAllocate)，UNLK (UNLinK) 命令は，ある領域を確保したり，解放したりするのに用いるもので，**LINK**命令はアドレス・レジスタAnをスタックに格納し，SP値をAnレジスタに転送し，最後にオペランドで指定したディスプレースメント値をSPに加算します．こうして，**LINK**命令によってサブルーチン実行に必要なパラメータ領域，ワーク領域がアロケートされ，そしてこれが**UNLK**命令によって解放されます．

(9) LEA命令

LEA命令は**Load Effective Address**で，実効アドレスをアドレス・レジスタAnにロードします（サイズはロング・ワードL）．

⑽　PEA命令

PEA命令は**Push Effective Address**で，実効アドレスをスタックにプッシュし，サイズは**L**となります．

次に，これらデータ転送命令を用いたプログラム例を考え，これのマシン語への変換の仕方を考えてみましょう．

3.2　データ転送命令のマシン語プログラミング例

例題7　次のプログラムをマシン語に変換しなさい．

```
        ORG        $3000
        MOVE.B     CNT1, D0
        MOVE.W     CNT2, D1
        MOVE.L     CNT3, D2
        MOVEA.L    DEST1, A0
        MOVEA.L    DEST2, A1
        MOVEA.L    DEST3, A2
        MOVE.B     D0, (A0 )
        MOVE.W     D1, 1(A1)
        MOVE.L     D2, 1(A2, D0)
CNT1    DC.B       $10
CNT2    DC.W       $2000
CNT3    DC.L       $30000000
DEST1   DC.L       $40000000
DEST2   DC.L       $50000000
DEST3   DC.L       $60000000
        END
```

解き方

MOVE命令のオペレーション・ワードのマシン語フォーマットは，次のように，

15 14	13 12	11　9 8　6	5　3 2　0
0　0	サイズ	デスティネーション (レジスタ)　(モード)	ソース (モード)　(レジスタ)

となります．

MOVE.B　CNT1, D0命令は，サイズ・フィールドは，バイト・オペレーションで▼01▼，デスティネーション・レジスタはD０で▼000▼，モードはデータ・レジスタ直接のアドレッシング・モードですから，▼000▼となります．ソースはCNT１でアブソリュート・ロングのアドレッシング・モードをとり，モード・フィールド＝▼111▼，レジスタ・フィールドを▼001▼にセットします．以上をマシン語フォーマットに代入して，

15 14	13 12	11　9 8　6	5　3 2　0	
0　0	0　1	0　0　0　0　0　0	1　1　1　0　0　1	………1039H

1039Hがオペレーション・ワードとなります．

次に，ソース実効アドレス・オペランドの２ワードが続き，この値はCNT１のアドレスで**0000302EH**となりますから，**MOVE.B　CNT1, D0**のマシン語は，

10390000302EH

となります．

MOVE.W　CNT2, D1は，同じマシン語フォーマットにサイズ＝ワード＝▼11▼，デスティネーション・レジスタ＝D１＝▼001▼，モード＝データ・レジスタ直接＝▼000▼，ソースはCNT２でアブソリュート・ロングですから，モード＝▼111▼，レジスタ＝▼001▼となり，以上をまとめて代入すると，

15 14	13 12	11　9 8　6	5　3 2　0	
0　0	1　1	0　0　1　0　0　0	1　1　1　0　0　1	………3239H

3239Hとなり，この次にソース実効アドレス・オペランドが続き，CNT２は▼00003030▼ですから，以上をまとめて，

323900003030H

が**MOVE.W　CNT2, D1**のマシン語となります．

まったく同様にして，**MOVE.L　CNT3,D2**のマシン語は**243900003032**Hとなります．

MOVEA.L　DEST1,A0命令は，アドレス転送**MOVEA**命令が用いられており，**MOVEA**命令のマシン語フォーマットは次のようになります．

15	14	13　12	11　　9	8　　6	5　　3	2　　0
0	0	サイズ	デスティネーション（レジスタ）	0　0　1	ソース（モード）	（レジスタ）

サイズ・フィールドはロング・ワードLで▼10▼をセットし，デスティネーション・レジスタはA０で▼000▼，ソースはアブソリュート・ロングのアドレッシング・モードとなり，モード・フィールド＝▼111▼，レジスタ・フィールド＝▼001▼となり，これらを代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	………2079H

2079Hとなります．

次に，ソース実効アドレス・オペラントが続き，DEST1は▼00003036▼ですから，**MOVEA.L　DEST1,A0**のマシン語は，

　　207900003036H

となります．

　　MOVEA.L　DEST2,A1　命令

　　MOVEA.L　DEST3,A2　命令

もまったく同様にして

　　22790000303AH

　　24790000303EH

のマシン語となることがわかります．

　　MOVE.B　D0,（A0）

　　MOVE.W　D1,１（A1）

　　MOVE.L　D2,１（A2,D0）

は，おのおの前の例題と同じようにマシン語に変換すると，次のようになります．

　　1080

33410001

25820001

以上をまとめると**表3.2**のようになります.

表3.2 例題7のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	10390000302E		MOVE.B	CNT1, D0
003006	323900003030		MOVE.W	CNT2, D1
00300C	243900003032		MOVE.L	CNT3, D2
003012	207900003036		MOVEA.L	DEST1, A0
003018	22790000303A		MOVEA.L	DEST2, A1
00301E	24790000303E		MOVEA.L	DEST3, A2
003024	1080		MOVE.B	D0, (A0)
003026	33410001		MOVE.W	D1, 1(A1)
00302A	25820001		MOVE.L	D2, 1(A2, D0)
00302E	10	CNT1	DC.B	$10
003030	2000	CNT2	DC.W	$2000
003032	30000000	CNT3	DC.L	$30000000
003036	40000000	DEST1	DC.L	$40000000
00303A	50000000	DEST2	DC.L	$50000000
00303E	60000000	DEST3	DC.L	$60000000
			END	

3.3 加算，減算命令

3.3.1 加算命令のマシン語

加算命令にはADD, ADDA, ADDI, ADDQ, ADDXがあります.

(1) ADD命令

ADD命令はソース・オペランドとデスティネーション・オペランドを加算し，その結果をデスティネーション・オペランドに格納します．オペレーション・サイズはバイト，ワード，ロング・ワードが可能で，マシン語フォーマット中のオペレーション・モード（OPモード）フィールドで指定します．

レジスタにはデータ・レジスタを指定することができ，そのデータ・レジスタNo.(番号)をレジスタ・フィールドで指定します．マシン語フォーマットを次に示します．

15	14	13	12	11　　9	8　　6	5　　0
1	1	0	1	レジスタ	OPモード	実効アドレス

レジスタ・フィールドの3ビットで，前述したようにD0からD7のデータ・レジスタのうち1つを指定し，OPモード・フィールドの3ビットでオペレーションのサイズとデスティネーション・オペランドの指定を行ないます．

オペレーション・モード

B(バイト)	W(ワード)	L(ロング・ワード)	オペレーション
0 0 0	0 0 1	0 1 0	Dn+EA→Dn
1 0 0	1 0 1	1 1 0	EA+Dn→EA

(2)　**ADDA命令**

ADDAはソース・オペランドとデスティネーション・オペランドを加算し，その結果をデスティネーションのアドレス・レジスタに格納します．デスティネーション・レジスタには，必ずアドレス・レジスタが使われます．オペレーション・サイズは，ワードかロング・ワードで，バイトは使えません．

次に，**ADDA**命令のマシン語フォーマットを示します．

15	14	13	12	11　　9	8　　6	5　　0
1	1	0	1	レジスタ	OPモード	実効アドレス

レジスタ・フィールド（3ビット）でアドレス・レジスタのNo.（番号）を指定しますが，**ADDA**命令においてはデスティネーションには必ずアドレス・レジスタがきます．

OPモード・フィールドでオペレーション・サイズが決められます（**ADD**命令では同時にデスティネーションの指定も行なわれましたが，**ADDA**命令では，アドレス・レジスタが必ず用いられるのは前述したとおりです）．

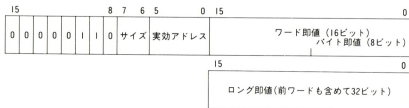
オペレーション・モード

W(ワード)	L(ロング・ワード)	オペレーション
0 1 1	1 1 1	An＋EA→An

実効アドレス・フィールドでソースのアドレッシング・モードの指定を行ないます．

(3) ADDI命令

ADDI命令は即値データをデスティネーション・オペランドと加算し，これをデスティネーションへ格納します．オペレーション・サイズはバイト，ワード，ロング・ワードが可能で，マシン語フォーマット中のサイズ・フィールドで指定します．

即値フィールドが実効アドレス・フィールドの次にきて，ここに即値が格納されます．即値はサイズ・フィールドの値によってバイト，ワード，ロング・ワードが指定されますが，バイト・サイズのときには即値ワード・フィールドの下位８ビットに格納されます．ADDI命令のマシン語フォーマットを次に示します．

サイズ・フィールド

　００……バイト・オペレーション

　０１……ワード・オペレーション

　１０……ロング・ワード・オペレーション

実効アドレス・フィールドは，デスティネーションのアドレッシング・モードを指定し，次の即値データ・フィールドに即値が入ります．サイズ＝▼00▼のときはバイト・オペレーションで，即値データは８ビット（１バイト）がバイト即値のところに格納されます．

サイズ＝▼01▼のときはワード・オペレーションで，即値データは１ワードがワード即値（16ビット）のフィールドに格納されます．サイズ＝▼10▼のと

きはロング・ワード・オペレーションで，即値データはロング・ワード（2ワード）が，ロング即値のフィールドに格納されます．

(4)　**ADDQ命令**

ADDQ命令は（ADD Quick）の意味で，即値データが1から8の範囲にあるとき，この命令実行を高速に行なえます．すなわち，実行クロック・サイクル数が少なく，かつコード・メモリのバイト数が少なくなる長所を持っている以外は，基本的にADDI命令と同一です．

1～8のクイック即値データをデスティネーション・オペランドに加算します．オペレーション・サイズはバイト，ワード，ロング・ワードが指定でき，アドレス・レジスタに対するワード，ロング・ワード・オペレーションもできます．

次に，ADDQ命令のマシン語フォーマットを示します．

15	14	13	12	11　　9	8	7　6	5　　　0
0	1	0	1	データ	0	サイズ	実効アドレス

サイズ・フィールド

0 0……バイト・オペレーション

0 1……ワード・オペレーション

1 0……ロング・ワード・オペレーション

データ・フィールド

3ビットで即値を表現．0は8を，それ以外の1から7はそのまま1から7を表現します．

実効アドレス・フィールドは，デスティネーション・オペランドのアドレッシング・モードの指定に用いられます．

(5)　**ADDX命令**

ADDX命令は，ソース・オペランドとデスティネーション・オペランドとさらにコンディション・コード中のXフラグを加算し，その結果をデスティネーションに格納します．

(ソース)＋(デスティネーション)＋Xフラグ→デスティネーション

ソース，デスティネーションのオペランドのアドレッシングは，データ・レジスタを直接用いるものと，アドレス・レジスタを用いたプリ・デクリメント・アドレッシング・モードでアドレスされるものとがあり，この選択はマシン語

フォーマット中のR／Mビットによって行なわれます。

この命令は倍精度，3倍精度といったマルチ・プレシジョン加算に用いられ，8086ではADC命令（すなわちキャリー付加算）に相当するものです。

R／Mビットが▼0▼でレジスター レジスタ，▼1▼でメモリーメモリのオペレーションとなり，オペレーション・サイズはバイト，ワード，ロング・ワードを指定することができます。

Xビットは C（キャリーフラグ）と同様に変化し，キャリーフラグと同じ値を保持していますから，Xビットの加算は，Cビットの加算と同じことになります。次に，**ADDX**のマシン語フォーマットを示します。

15	14	13	12	11 　9	8	7 　6	5		3	2 　0
1	1	0	1	DEST. レジスタ (Rx)	1	サイズ	0	0	R/M	SRC レジスタ (Ry)

DEST.レジスタ（Rx）フィールドは，デスティネーション・レジスタのNo.を指定するフィールドで，R／M＝0のときはレジスタとなりますから，データ・レジスタのNo. を選択することになり，R／M＝1のときはメモリ参照となり，プリ・デクリメント・アドレッシング・モードで使うアドレス・レジスタのNo.を選択するのに用いられることになります。

サイズ・フィールドはオペレーションのサイズを指定するもので，

サイズ・フィールド

0 0……バイト・オペレーション

0 1……ワード・オペレーション

1 0……ロング・ワード・オペレーション

となります。

R／Mフィールドは，オペランドのアドレッシング・モードの選択に用いられ，

0……データ・レジスタからデータ・レジスタ

1……メモリからメモリ

となります。

SRCレジスタ(Ry)フィールドは，ソース・レジスタのNo.を指定し，R／M＝0であればデータ・レジスタNo. を，またR／M＝1のときはメモリ参照ですから，プリ・デクリメント・アドレッシング・モードで使用するアドレス・レジスタのNo.を指定します。

3．3．2　減算命令のマシン語

減算命令にはSUB，SUBA，SUBI，SUBQ，SUBXの各命令が用意されており，加算が減算になった点を除いてADDの場合と同じです．すなわち，通常のデータ・レジスタを用いる減算はSUB，アドレス・レジスタからの減算はSUBA，即値の減算はSUBI，クイック即値（１～８）の減算はSUBQ，Xフラグをも含めた減算はSUBXとなります．

(1)　SUB命令

SUB命令はデスティネーションからソース・オペランドを減算し，その結果をデスティネーションに格納します．マシン語フォーマット中のOPモードのビットによって，オペレーションのサイズ（B，W，L）とデスティネーションがどちらかを（すなわちDnかEAかを）指定します．

次にマシン語フォーマットを示します．

15	14	13	12	11　　9	8　　6	5　　　　0
1	0	0	1	レジスタ	OPモード	実効アドレス

オペレーション・モード

B(バイト)	W(ワード)	L(ロング・ワード)	オペレーション
0 0 0	0 0 1	0 1 0	Dn－EA→Dn
1 0 0	1 0 1	1 1 0	EA－Dn→EA

レジスタ・フィールドの３ビットでデータ・レジスタ（D０～D７）のNo.を指定し，OPモードで，上に示したように，オペレーションのサイズとデスティネーションがDnかEAかの指定を行ないます．また，実効アドレス・フィールドでアドレッシング・モードの指定が行なわれます．

(2)　SUBA命令

SUBA命令はデスティネーションのアドレス・レジスタからソース・オペランドを減算し，その結果をアドレス・レジスタに格納します．オペレーションのサイズは，ワードかロング・ワードです．

次にマシン語フォーマットを示します．

レジスタ・フィールドでアドレス・レジスタのNo．を指定し，これは常にデスティネーションとなります．オペレーション・モード・フィールドによってW（ワード），L（ロング・ワード）の指定が行なわれ，

0 1 1……ワード・オペレーション

1 1 1……ロング・ワード・オペレーション

となります．また，実効アドレス・フィールドによってソースのアドレッシング・モードを指定します．

(3)　**SUBI，SUBQ，SUBX**

SUBI，**SUBQ**は即値減算，クイック即値減算で，**SUBX**はXフラグ付の減算で，8086では**SBB**に相当する命令です．

3.4　加減算命令のマシン語プログラミング例

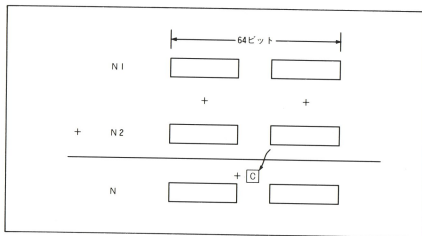



図3．1　64ビットの多倍精度加算

例題8　次の多倍精度(64ビット長)加算のプログラムをマシン語に変換しなさい(ハンド・アセンブルせよ).

```
        ORG      $3000
N1      DC.L     $12345678
        DC.L     $90ABCDEF
N2      DC.L     $00000008
        DC.L     $0
N       DC.L     $0
        DC.L     $0
        MOVE.L   N1, D1
        MOVE.L   N1+4, D0
        MOVE.L   N2, D3
        MOVE.L   N2+4, D2
        ADD.L    D2, D0
        ADDX.L   D3, D1
        MOVE.L   D0, N+4
        MOVE.L   D1, N
        TRAP     #13
        END
```

解き方

これは，**ADDX**命令と**ADD**命令を用いて，64ビットの多倍精度加算を行なう例です．**図3.1**に示すように，N１（64ビット長）とN２（64ビット長）を加算して，その結果をNに格納します．下位32ビットの加算では**ADD**命令を使用し，上位32ビットの加算では，**ADDX**命令を使用します．

下位32ビットの加算によってキャリーが立った場合，**ADDX**命令によって，このキャリーの値が上位32ビット加算時に同時に加算され，多倍精度加算が実行されます．キャリーが下位32ビットの加算時に立たなかった場合は，**ADDX**命令実行時，同時にゼロ（=0）が加算されるのみです．このように，多倍精度加算プログラムでは最下位の数の加算にだけ**ADD**命令を使い，それ以外の(最上位までの)すべての数の加算には，**ADDX**命令を使います(**ADDX**命令は(ソース)+（デスティネーション)+（X）→（デスティネーション）です)．

それでは，例題8をマシン語に変換します．ORGは3000Hから開始していますから，MOVE.L　N1,D1命令は3018Hから始まります．

MOVE命令のマシン語フォーマットは，

15	14	13　12	11　9	8　6	5　3	2　0
0	0	サイズ	デスティネーション (レジスタ)	(モード)	ソース (モード)	(レジスタ)

で，サイズ，デスティネーション，ソースの各フィールドに値をセットしていきます．

サイズ・フィールドはMOVE.LのLよりロング・ワード・オペレーションですから▼10▼をセットします．

デスティネーションはD1で，アドレッシング・モードはデータ・レジスタ直接，したがって，モード・フィールドは表3.3（これは22ページに示した表2.1と同じものですが，大変重要なものですので，もう一度掲載しておきます）より▼000▼，レジスタ・フィールドは▼001▼となります．

ソースはN1で，アブソリュート（絶対）ショートのアドレッシング・モードですから，モード・フィールドは▼111▼，レジスタ・フィールドは▼000▼となります．

以上をMOVE命令のマシン語フォーマットに代入すると，

表3.3　モード，レジスタ・フィールドによって決められるアドレッシング・モード

モード・フィールド	レジスタ・フィールド	アドレッシング・モード
0 0 0	Dn	データ・レジスタ直接
0 0 1	An	アドレス・レジスタ直接
0 1 0	An	アドレス・レジスタ間接
0 1 1	An	ポスト・インクリメント・アドレス・レジスタ間接
1 0 0	An	プリ・デクリメント・アドレス・レジスタ間接
1 0 1	An	ディスプレースメント付アドレス・レジスタ間接
1 1 0	An	インデックス，ディスプレースメント付アドレス・レジスタ間接
1 1 1	0 0 0	アブソリュート・ショート
1 1 1	0 0 1	アブソリュート・ロング
1 1 1	0 1 0	ディスプレースメント付プログラム・カウンタ相対
1 1 1	0 1 1	インデックス，ディスプレースメント付プログラム・カウンタ相対
1 1 1	1 0 0	即　値

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	1	0	0	0	1	1	1	0	0	0	………2238H

となり，これがマシン語の第１ワードとなります．これに続いて，ソース実効アドレスのオペランドがきますが，Ｎ１のアドレスは▼3000▼ですから，以上をまとめると，

22383000H

これが**MOVE.L　N1,D1**のマシン語となります．

MOVE.L　N1+4,D0は同じマシン語フォーマットにおいて，サイズ＝ロング＝▼10▼，デスティネーションのモード＝データ・レジスタ直接＝▼000▼，レジスタ＝D0＝▼000▼，ソースはアブソリュート・ショートで，モード＝▼111▼，レジスタ・フィールド＝▼000▼となり，これらを代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	………2038H

となります．

次に，ソース実効アドレスが続き，N1+4は，▼3004▼ですから，以上をまとめると**MOVE.L　N1+4,D0**のマシン語は，

20383004H

となります．

MOVE.L　N2,D3は同じマシン語のフォーマットにおいて，サイズ＝▼01▼，デスティネーションのモード＝▼000▼，レジスタ＝▼011▼，ソースはモード＝▼111▼，レジスタ＝▼000▼を代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	………2638H

となり，ソース実効アドレスの▼3008▼を加えて，**MOVE.L　N2,D3**のマシン語は，

26383008H

となります．

同様にして，**MOVE.L　N2+4,D2**は同じマシン語フォーマットにサイズ=▼10▼，デスティネーションのモード=▼000▼，レジスタ=▼010▼，ソースのモード=▼111▼，レジスタ=▼000▼を代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	0	0	1	0	0	0	0	1	1	1	0	0	0	………2438H

となり，ソース実効アドレスの▼300C▼を加えて，**MOVE.L　N2+4,D2**のマシン語は，

2438300CH

となります．

ADD.L　D2,D0のマシン語フォーマットは，次に示すとおりです．

15	14	13	12	11 ～ 9	8 ～ 6	5 ～ 0
1	1	0	1	レジスタ	OPモード	実効アドレス

ここにおいて，レジスタ・フィールドはデータ・レジスタのNo.を指定するフィールドで，オペレーション・モード・フィールドは，この加算オペレーションのモードを指定するフィールドです．オペレーションのサイズとデスティネーション・オペランドは，このオペレーション・モード・フィールドによって決められます．

オペレーション・モード

B(バイト)	W(ワード)	L(ロング・ワード)	オペレーション
0 0 0	0 0 1	0 1 0	(<Dn>)+(<EA>)→<Dn>
1 0 0	1 0 1	1 1 0	(<EA>)+(<Dn>)→<EA>

実効アドレス・フィールドは，〈EA〉のアドレッシング・モードを指定するフィールドとして用いられます．

ADD.L　D2，D0でデスティネーション・オペランドはD0ですから，OPモードは▼010▼となり，レジスタ・フィールドは▼000▼となります．

実効アドレス・フィールドで〈EA〉のアドレッシング・モードを指定しますが，D2ですから，データ・レジスタ直接アドレッシング・モードで，その値は

▼000010▼となります。

これらの値をマシン語フォーマットに代入して，

D082HがADD.L　D2,D0命令のマシン語となります。

ADDX.L　D3,D1のマシン語フォーマットを次に示します。

15	14	13	12	11〜9	8	7〜6	5	4	3	2〜0
1	1	0	1	DESTレジスタ (Rx)	1	サイズ	0	0	R/M	SRCレジスタ (Ry)

ここで，おのおののフィールドの意味をもう一度復習してみましょう。

DESTレジスタ（Rx）フィールドは，デスティネーション・レジスタのNo.を指定し，R／M＝0のときはレジスタですから，データ・レジスタのNo.を選択し，R／M＝1のときはメモリですから，プリ・デクリメント・アドレッシング・モードで使用するアドレス・レジスタのNo.を選択します。

サイズ・フィールドは他の命令の場合とまったく同様に，オペレーションのサイズを指定するもので，▼00▼でバイト・オペレーション，▼01▼でワード・オペレーション，▼10▼でロング・ワード・オペレーションとなります。

R／Mのフィールドはオペランドのアドレッシング・モードの選択に使用し，

0……データ・レジスタからデータ・レジスタ

1……メモリからメモリ

の2つのうち，どちらか1つが選択されます。

SRCレジスタ（Ry）フィールドはソース・レジスタのNo.を指定し，R／M＝0であればデータ・レジスタNo.を，またR／M＝1であれば，メモリ参照ですから，プリ・デクリメント・アドレッシング・モードで使用するアドレス・レジスタのNo.を指定します。

ADDX.L　D3，D1命令でSRCはデータ・レジスタD3，デスティネーションDESTはデータ・レジスタD1ですから，マシン語フォーマット中のDESTレジスタ・フィールドは▼001▼，SRCレジスタ・フィールドは▼011▼，R／Mフィールドはデータ・レジスタとデータ・レジスタとのキャリー付加算ですから，R／M＝0，オペレーションのサイズはロング・ワードですからサイズ＝▼10▼，

これらをマシン語フォーマットに代入して，

15	14	13	12	11		9	8	7	6	5	4	3	2		0
1	1	0	1	0	0	1	1	1	0	0	0	0	0	1	1

………D383H

D383Hが**ADDX.L　D3,D1**のマシン語となります．

MOVE.L　D0,N+4のマシン語は，**MOVE**命令のマシン語フォーマットに，サイズ＝ロング・ワード＝▼10▼，デスティネーションのレジスタ・フィールド＝アブソリュート・ショート＝▼000▼，モード・フィールド＝▼111▼となり，ソースのモード・フィールド＝D0＝直接アドレッシング＝▼000▼，レジスタ・フィールド＝▼000▼を代入して，

となり，これにデスティネーション(DEST)実効アドレス・ワードが続き，N+4の実効アドレスは▼3014▼ですから，以上をまとめて，

21C03014H

が**MOVE.L　D0,N+4**のマシン語となります．

これとまったく同様にして，

MOVE.L　D1,Nのマシン語は，

21C13010H

となります．

TRAPのマシン語フォーマットは，

15	14	13	12	11	10	9	8	7	6	5	4	3　　0
0	1	0	0	1	1	1	0	0	1	0	0	トラップ・ベクタ番号

で，トラップ・ベクタ番号フィールドに#13を代入して，**TRAP　#13**のマシン語は，

4E4DH

となります．

以上の命令をすべてまとめると，**表3.4**のようなハンド・アセンブルしたリス

表3．4　例題8のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	12345678	N1	DC.L	$12345678
003004	90ABCDEF		DC.L	$90ABCDEF
003008	00000008	N2	DC.L	$00000008
00300C	00000000		DC.L	$0
003010	00000000	N	DC.L	$0
003014	00000000		DC.L	$0
003018	22383000		MOVE.L	N1, D1
00301C	20383004		MOVE.L	N1+4, D0
003020	26383008		MOVE.L	N2, D3
003024	2438300C		MOVE.L	N2+4, D2
003028	D082		ADD.L	D2, D0
00302A	D383		ADDX.L	D3, D1
00302C	21C03014		MOVE.L	D0, N+4
003030	21C13010		MOVE.L	D1, N
003034	4E4D		TRAP	#13
			END	

トが得られます．

例題9

次のプログラムをマシン語に変換しなさい．

```
          ORG       $3000
          MOVEA.L   SRCPTR, A0
          MOVEA.L   DESTPTR, A1
          MOVE.L    MPNO, D0
          MOVE      #$4, CCR
ABC1      ADDX      -(A0), -(A1)
          DBRA      D0, ABC1
          TRAP      #13
SRCPTR    DC.L      $1000
DESTPTR   DC.L      $1010
MPNO      DC.L      $4
          END
```

解き方

MOVEA.L　SRCPTR，A0のマシン語フォーマットは，

15	14	13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0	サイズ	デスティネーション (レジスタ)	0 0 1	ソース (モード)	(レジスタ)

で，オペレーション・サイズ＝ロング・ワード＝▼10▼，デスティネーション・レジスタ＝A0＝▼000▼，ソースのモード・フィールド＝アブソリュート・ロング＝▼111▼，レジスタ・フィールドもアブソリュート・ロングですから，レジスタ・フィールド＝▼001▼，これらをマシン語フォーマットに代入すると，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	0	1	1	1	1	0	0	1	………2079H

となり，この次にソース実効アドレス・ワードが続き，**SRCPTR**のアドレスは▼0000301E▼ですから(各命令の占めるバイト数を計算する)，以上をまとめて，

20790000301EH

が**MOVEA.L　SRCPTR，A0**のマシン語となります．

MOVEA.L　DESTPTR，A1のマシン語もこれと同様にして，

227900003022H

となります．

MOVE.L　MPNO，D0のマシン語も何度も行なったように，

203900003026H

となります．

MOVE　#$4,CCRのマシン語フォーマットは，

15	14	13	12	11	10	9	8	7	6	5 … 0
0	1	0	0	0	1	0	0	1	1	実効アドレス

で，実効アドレス・フィールドでソースのアドレッシング・モードを指定します．ここでは#$4で即値のアドレッシング・モードですからモード＝▼111▼，

レジスタ・フィールド＝▼100▼となり，これらをマシン語フォーマットに代入して，

44FCHが**MOVE　#$4,CCR**のマシン語の第1ワードで，次に即値フィールドが続き，これは▼0004▼ですから，以上をまとめると，

44FC0004H

となり，**44FC0004H**が**MOVE　#$4,CCR**のマシン語となります．

ADDX　−(A0)，−(A1)はマシン語フォーマット；

15	14	13	12	11　　9	8	7　6	5	4	3	2　　0
1	1	0	1	DESTレジスタ (Rx)	1	サイズ	0	0	R/M	SRCレジスタ (Ry)

において，メモリーメモリですからR／M＝▼1▼，サイズ＝ワード＝▼01▼，DESTレジスタ・フィールド＝A1＝▼001▼，SRCレジスタ・フィールド＝A0＝▼000▼で，これらをマシン語フォーマットに代入して，

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	0	1	0	0	1	1	0	1	0	0	1	0	0	0	………D348H

D348Hが**ADDX　−(A0)，−(A1)**のマシン語となります．

DBRA　D0,ABC1のマシン語フォーマット（＝**DBcc**）は，

15	14	13	12	11　　8	7	6	5	4	3	2　　0	15　　0
0	1	0	1	条　件	1	1	0	0	1	レジスタ	ディスプレースメント

で，条件フィールドの条件が満足されると，オペランド先へジャンプしないで次の命令を実行します．条件が満足されないと，カウンタのデータ・レジスタ内容を−1して，この値が−1になると次の命令を実行します．

ディスプレースメントは，この命令とラベルまでの変位（バイト）であり，ここでは▼FFFC▼となり，条件には▼0001▼　（＝常に偽）（＝never true）を

表3.5 例題9のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	20790000301E		MOVEA.L	SRCPTR, A0
003006	227900003022		MOVEA.L	DESTPTR, A1
00300C	203900003026		MOVE.L	MPNO, D0
003012	44FC0004		MOVE	#$4, CCR
003016	D348	ABC1	ADDX	−(A0), −(A1)
003018	51C8FFFC		DBRA	D0, ABC1
00301C	4E4D		TRAP	#13
00301E	00001000	SRCPTR	DC.L	$1000
003022	00001010	DESTPTR	DC.L	$1010
003026	00000004	MPNO	DC.L	$4
			END	

用いて，これらを代入して，

51C8FFFCH

がマシン語となります．

以上をまとめると**表3.5**のようなハンド・アセンブル・リストが得られます．

3.5 乗算，除算命令

3.5.1 乗算命令のマシン語

乗算命令には，符号付乗算命令と符号なし乗算命令とがあり，ともにデータ・レジスタDnの下位16ビットとソース・オペランドの16ビットとを乗算し，その32ビットの積をデータ・レジスタDnにセットします．

符号付乗算命令は**MULS** (**Signed Multiply**)，符号なし乗算命令は**MULU** (**Unsigned Multiply**) です．

MULU命令は，符号なしの整数（16ビット）と整数の乗算で，結果は32ビットの整数となります．

MULU 〈EA〉, Dn……… 〈EA〉×Dn→Dn

ソース・オペランドとデスティネーション・オペランドを符号なし乗算し，結果をデスティネーション・オペランドに格納します．デスティネーション・

オペランドは，必ずデータ・レジスタDnがきて，ここに32ビット長で積がセットされます．ソース・オペランドには“アドレス・レジスタ直接”を除く他のすべてのアドレッシング・モードを指定することができます．

MULU命令のマシン語フォーマットは，

15	14	13	12	11　　9	8	7	6	5　　　　0
1	1	0	0	レジスタ	0	1	1	実効アドレス

で，レジスタ・フィールドはデスティネーションのデータ・レジスタDnのNo.を指定し，実効アドレス・フィールドはソース・オペランドのアドレッシング・モードを指定します．

コンディション・コードは，演算結果によって次のように変化します．

X	N	Z	V	C
−	*	*	0	0

Nフラグは演算結果の最上位ビット（MSB）が1であればセットされ，そうでなければクリアされます．Zフラグは，演算結果がゼロであればセットされ，そうでなければクリアされます．

すなわち，ZフラグとNフラグは，演算結果によってセットまたはクリアされます．VフラグとCフラグは常にクリアされ，またXフラグは変化しません．

MULS命令は，符号付きの整数（16ビット）と整数の乗算で，結果は32ビットの符号付き整数となります．このように，符号付きで乗算が行なわれ，結果はデスティネーション・オペランドに格納されます．

デスティネーション・オペランドは，必ずデータ・レジスタDnが用いられ，ここに32ビット長の符号付き整数がセットされます．ソース・オペランドは“アドレス・レジスタ直接”を除く他のすべてのアドレッシング・モードが指定できます．

MULS 〈EA〉, Dn……… 〈EA〉×Dn→Dn

MULS命令のマシン語フォーマットは，次のようになります．

15	14	13	12	11 9	8	7	6	5 0
1	1	0	0	レジスタ	1	1	1	実効アドレス

MULU命令の場合と同様に，レジスタ・フィールドでデスティネーションのデータ・レジスタDnのNo.を指定し，実効アドレス・フィールドでソース・オペランドのアドレッシング・モードを指定します．コンディション・コードの変化は**MULU**命令と同様です．

3.5.2　除算命令のマシン語

除算命令には，符号付き除算命令と符号なし除算命令とがあり，データ・レジスタDnの32ビットの数をソース・オペランドの16ビットの数で除算し，商と余りをデータ・レジスタにセットします．商は下位16ビット，すなわち下位ワードにセットされ，余りは上位16ビットすなわち上位ワードにセットされます．

符号付き除算命令は**DIVS** (**Signed　Divide**)，符号なし除算命令は**DIVU** (**Unsigned Divide**) です．

DIVU,DIVSともに除算命令の実行で，次のような特別な動作をすることがあります．

- 0 (ゼロ)で除算をするとトラップが発生し，例外処理が開始される．
- オーバーフローが命令の完了以前に発生し，これが検出されるとオーバーフローフラグ(Vフラグ)がセットされ，命令は実行されないままとなり，そして，第2オペランド内容は変化しない．

DIVU命令は，符号なしの数として除算が行なわれ，商と余りの除算結果はデータ・レジスタDnにセットされます．

DIVU　<EA>，Dn………Dn÷ <EA> →Dn

(Dn（上位ワード）…余り，Dn（下位ワード）…商)

デスティネーション・オペランドをソース・オペランドで除算して，その結果をデスティネーションに格納します．デスティネーション・オペランドは32ビット長（ロング・ワード）が用いられ，ソース・オペランドは16ビット長（ワード）のデータが用いられます．符号なし演算で除算は行なわれ，結果はデスティネーション・オペランドのデータ・レジスタDnにセットされ，上位ワードに余りが，下位ワードに商がセットされます．

DIVU命令のマシン語フォーマットは，

15	14	13	12	11　9	8	7	6	5　0
1	0	0	0	レジスタ	0	1	1	実効アドレス

でレジスタ・フィールドは，デスティネーション・オペランドのデータ・レジスタDnのNo.を指定し，実効アドレス・フィールドは，ソース・オペランドのアドレッシング・モードを指定します．このアドレッシング・モードには，“アドレス・レジスタ直接”を除く他のすべてのモードが指定できます．

コンディション・コードは次のようになります．

X	N	Z	V	C
−	*	*	*	0

Nフラグは結果の商の最上位ビット（MSB）が1であればセットされ，そうでなければクリアされます．オーバーフローが生じた場合は不定となります．

Zフラグは商がゼロ（0）であればセットされ，そうでなければクリアされます．ただし，オーバーフローが生じた場合は不定となります．Vフラグはオーバーフローが検出されたときセットされ，そうでなければクリアされます．Cフラグは常にクリアされ，Xフラグは変化しません．

DIVS命令は符号付きの数として除算オペレーションが行なわれ，商と余りの除算結果はデータ・レジスタDnにセットされます．

DIVS　〈EA〉，Dn………**Dn÷〈EA〉→Dn**

（Dn（上位ワード）…余り，Dn（下位ワード）…商）

符号付きで除算が行なわれる点を除いて，他は**DIVU**命令と同様です．

DIVS命令のマシン語フォーマットは，次のようになります．

15	14	13	12	11　9	8	7	6	5　0
1	0	0	0	レジスタ	1	1	1	実効アドレス

DIVU命令の場合と同様に，レジスタ・フィールドでデスティネーションのデータ・レジスタDnのNo.を指定し，実効アドレス・フィールドでソース・オペランドのアドレッシング・モード（“アドレス・レジスタ直接”以外の）を指定します．コンディション・コードの変化は**DIVU**命令と同様です．

次に，乗算，除算命令を実際にマシン語へ変換する例題を解いてみましょう．

3.6 乗除算命令のマシン語プログラミング例

例題10

次のプログラムをマシン語に変換しなさい(ハンド・アセンブルせよ)．

```
        ORG       $3000
        MOVE.W    N1, D1
        MOVE.W    N2, D0
        MULU      D1, D0
        DIVU      #10, D0
        MOVE.W    D0, QU
        SWAP      D0
        MOVE.W    D0, RM
        TRAP      #13
N1      DC.W      $3
N2      DC.W      $4
QU      DC.W      0
RM      DC.W      0
        END
```

解き方

MOVE命令のマシン語フォーマットは，

15	14	13 12	11　9 8　6	5　3 2　0
0	0	サイズ	デスティネーション (レジスタ) (モード)	ソース (モード) (レジスタ)

で，サイズ，デスティネーション，ソースの各フィールドに値をセットしていきます．

サイズ・フィールドは，**MOVE.W**のWよりワード・オペレーションですから

▼11▼をセットします．

MOVE.W　N1,D1命令のデスティネーションはD1で，アドレッシング・モードはデータ・レジスタ直接．したがって，モード・フィールドは▼000▼，レジスタ・フィールドは▼001▼となります．

ソースはN1で，アブソリュート(絶対)・ロングのアドレッシング・モードになり，モード・フィールドは▼111▼，レジスタ・フィールドは▼001▼となります．

以上を**MOVE**命令のマシン語フォーマットに代入すると，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	1	0	0	0	1	1	1	0	0	1	………3239H

となり，これがマシン語の第1ワードとなります．これに続いてソース実効アドレスのオペランド・ワードがきますが，N1のアドレスは▼00003022▼ですから，以上をまとめると，

323900003022H

となり，これが**MOVE.W　N1,D1**のマシン語となります．

MOVE.W　N2,D0は同じマシン語フォーマットにおいて，サイズ＝ワード＝▼11▼，デスティネーションのモード＝データ・レジスタ直接＝▼000▼，レジスタ＝D0＝▼000▼，ソースはアブソリュート・ロングで，モード＝▼111▼，レジスタ・フィールド＝▼001▼となり，これらを代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	1	………3039H

となります．

次に，ソース実効アドレスが続き，N2は▼00003024▼ですから，以上をまとめると**MOVE.W　N2,D0**のマシン語は，

303900003024H

となります．

乗算命令**MULU　D1,D0**をマシン語に変換します．符号なし乗算命令**MULU**のマシン語フォーマットは，

15	14	13	12	11　9	8	7	6	5　0
1	1	0	0	レジスタ	0	1	1	実効アドレス

です．

レジスタ・フィールドは，デスティネーション・オペランドのデータ・レジスタのNo.（番号）をセットするところで，D0レジスタが用いられますから，▼000▼をセットします．

実効アドレス・フィールドは，ソース・オペランドのアドレッシング・モードを指定するフィールドで，"データ・レジスタ直接"のアドレッシング・モードが使用されていますから，実効アドレス＝▼000001▼となります．

以上を**MULU**のマシン語フォーマットに代入すると，

となり，**C0C1H**が**MULU　D1,D0**のマシン語となります．

次に，除算命令**DIVU　#10,D0**をマシン語に変換します．符号なし除算命令**DIVU**のマシン語フォーマットは次のとおりです．

レジスタ・フィールドは，デスティネーション・オペランドのデータ・レジスタNo.＝▼000▼がセットされ，実効アドレス・フィールドは，ソース・オペランドのアドレッシング・モード＝即値モード＝▼111100▼がセットされて，以上から，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	………80FCH

80FCHが**DIVU　#10,D0**命令のマシン語の第1ワードとなります．これに続いてソース実効アドレス・オペランドがきますが，#10の即値ですから，以上をま

とめて,

80FC000AH

これが**DIVU　#10,D0**のマシン語となります.

MOVE.W　D0,QUのマシン語は,**MOVE**命令のマシン語フォーマットに,サイズ=ワード=▼11▼,デスティネーションのレジスタ・フィールド=アブソリュート・ロング=▼001▼,モード・フィールド=▼111▼となり,ソースのモード・フィールド=D0=直接アドレッシング=▼000▼,レジスタ・フィールド=▼000▼を代入して,

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	………33C0H

となり,これにデスティネーション実効アドレス・ワードが続き,QUの実効アドレスは▼00003026▼ですから,以上をまとめて,

33C000003026H

が**MOVE.W　D0,QU**のマシン語となります.

これとまったく同様にして,**MOVE.W　D0,RM**のマシン語は,

33C000003028H

となります.

SWAPのマシン語フォーマットは,

15	14	13	12	11	10	9	8	7	6	5	4	3	2 – 0
0	1	0	0	1	0	0	0	0	1	0	0	0	レジスタ

で,**SWAP　D0**のマシン語はレジスタ・フィールドに▼000▼(=D0)を代入して,

4840H

となります.

TRAP　#13のマシン語は,**4E4DH**ですから,以上の命令をすべてまとめると,次のページの**表3.6**のようなハンド・アセンブルしたリストが得られます.

表3.6　例題10のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	323900003022		MOVE.W	N1, D1
003006	303900003024		MOVE.W	N2, D0
00300C	C0C1		MULU	D1, D0
00300E	80FC000A		DIVU	#10, D0
003012	33C000003026		MOVE.W	D0, QU
003018	4840		SWAP	D0
00301A	33C000003028		MOVE.W	D0, RM
003020	4E4D		TRAP	#13
003022	0003	N1	DC.W	$3
003024	0004	N2	DC.W	$4
003026	0000	QU	DC.W	0
003028	0000	RM	DC.W	0
			END	

3.7　比較命令

3.7.1　比較命令のマシン語

比較命令にはCMP, CMPA, CMPI, CMPMが用意されています．

(1)　**CMP (CoMPare)命令**

CMP命令は，ソースとデスティネーションの内容を比較し，その結果フラグのみ変化し，オペランドの内容はともに変化しません．デスティネーション・オペランドのデータ・レジスタDnからソース・オペランドを減算し，それによってコンディション・コードが影響を受け，変化します．

CMP <EA>, Dn …… Dn－<EA>→フラグのみ変化

オペレーション・サイズはバイト，ワード，ロング・ワードが可能で，マシン語フォーマットは次のとおりです．

15	14	13	12	11 〜 9	8 〜 6	5 〜 0
1	0	1	1	レジスタ	OPモード	実効アドレス

レジスタ・フィールドはデスティネーションのデータ・レジスタDnのNo.を指定し，実効アドレス・フィールドはソース・オペランドのアドレッシング・モードを指定し，オペレーション・モード・フィールドでオペレーション・サイズを指定します．

オペレーション・モード

0 0 0………………バイト・オペレーション

0 0 1………………ワード・オペレーション

0 1 0………………ロング・ワード・オペレーション

（アドレス・レジスタ直接のアドレッシング・モードのとき，バイト・オペレーションは不可）

(2) CMPA (CoMPare Address)命令

CMPA命令は，デスティネーション・オペランドのアドレス・レジスタAnからソース・オペランドを減算し，それによってコンディション・コードが変化します．デスティネーション・オペランドには，必ずアドレス・レジスタが使われます．

CMPA　〈EA〉, An …… An－〈EA〉→フラグのみ変化

オペレーション・サイズはワード，ロング・ワードが指定できます．次にCMPA命令のマシン語フォーマットを示します．

レジスタ・フィールドはデスティネーションのアドレス・レジスタAnのNo.を指定し，実効アドレス・フィールドはソース・オペランドのアドレッシング・モードを指定し，オペレーション・モード・フィールドでオペレーション・サイズを指定します．

オペレーション・モード

0 1 1……ワード・オペレーション（ソース・オペランド（ワード）は符号拡張されて，32ビット・アドレス・レジスタと比較）

1 1 1……ロング・ワード・オペレーション

(3) CMPI (CoMPare Immediate)命令

CMPI命令は，デスティネーション・オペランドから即値データを減算し，それによってコンディション・コードが変化します．しかし，デスティネーショ

ン・オペランドの内容は変わりません．

CMPI　#<データ>，<EA>……<EA>−#<データ>→フラグのみ変化

オペレーションサイズはバイト，ワード，ロング・ワードが指定でき，マシン語フォーマットの中のサイズ・フィールドで指定します．即値フィールドが実効アドレス・フィールドの次にきて，ここに即値データが格納されます．

即値データは，サイズ・フィールドのビット・パターンによってバイト，ワード，ロング・ワードが指定されますが，バイト・サイズのときには，ワード即値フィールドの下位バイトに格納されます．即値フィールドも含めてCMPI命令のマシン語フォーマットを**図3.2**に示します．

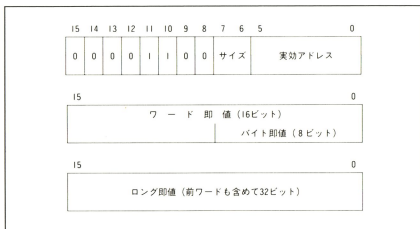



図3.2　CMPI命令のマシン語フォーマット

サイズ・フィールド

0 0…………バイト・オペレーション

0 1…………ワード・オペレーション

1 0…………ロング・ワード・オペレーション

上に示したようにサイズ・フィールドのビット・パターンによってオペレーションのサイズが決められます．

実効アドレス・フィールドは，デスティネーションのアドレッシング・モードを指定し，次の即値データ・フィールドに即値が入ります．サイズ＝▼00▼のときはバイト・オペレーションで，即値データは8ビット(1バイト)がバイト即値のところに格納されます．

サイズ＝▼01▼のときはワード・オペレーションで，即値データは１ワードがワード即値（16ビット）のフィールドに格納されます．

サイズ＝▼10▼のときはロング・ワード・オペレーションで，即値データはロング・ワード（２ワード）がロング即値のフィールドに格納されます．

(4)　CMPM（CoMPare Memory)命令

CMPM命令は，デスティネーション・オペランドからソース・オペランドを減算し，その結果によってコンディション・コードが変化します．もちろんオペランドの内容はともに変化しません．

ソース，デスティネーションの両方のオペランドとも，“ポスト・インクリメント・アドレス・レジスタ間接”アドレッシング・モードが用いられます．

CMPM　(Ay)＋，(Ax)＋　……　(Ax)－(Ay)→フラグのみ変化

オペレーション・サイズは，バイト，ワード，ロング・ワードが指定できます．次にCMPM命令のマシン語フォーマットを示します．

15	14	13	12	11　　9	8	7　6	5	4	3	2　　0
1	0	1	1	レジスタRx	1	サイズ	0	0	1	レジスタRy

レジスタRxフィールドは，デスティネーションのアドレス・レジスタ(ポスト・インクリメント・アドレス・レジスタ間接のアドレッシング・モードで用いられます）のNo.を指定するのに使用します．

サイズ・フィールドはオペレーション・サイズを指定し，次のようになります．

サイズ・フィールド

　００……バイト・オペレーション

　０１……ワード・オペレーション

　１０……ロング・ワード・オペレーション

レジスタRyフィールドは，ソース・オペランドのアドレス・レジスタ(これもポスト・インクリメント・アドレス・レジスタ間接のアドレッシング・モードで用いられます）のNo.を指定するのに用いられます．

それでは，次に比較命令のマシン語変換の例題を実際に行なってみましょう．

3.8　比較命令のマシン語プログラミング例

例題11　次のプログラムをマシン語に変換しなさい(ハンド・アセンブルせよ).

```
         ORG      $3000
         MOVEA.L  SRC01, A0
         MOVEA.L  SRC02, A1
         MOVEA.L  SRC03, A2
         MOVE.W   (A0), D0
         MOVE.W   (A1), D1
         CMP.W    D0, D1
         CMPA.L   #$1FFFF, A1
         CMPA.L   A2, A3
         CMPI.B   #$56, (A2)+
         CMPM.L   (A1)+, (A2)+
         TRAP     #13
SRC01    DC.L     $10000
SRC02    DC.L     $11000
SRC03    DC.L     $12000
         END
```

解き方

MOVEA.L　SRC01, A0命令は，アドレス転送**MOVEA**命令が用いられ，このマシン語フォーマットは，

15	14	13 12	11 9	8 6	5 3	2 0
0	0	サイズ	デスティネーション (レジスタ)	0 0 1	ソース (モード)	(レジスタ)

で，サイズはロング・ワードですから▼10▼，デスティネーション・レジスタはA0ですからレジスタ・フィールドは▼000▼，ソースはアブソリュート・ロングのアドレッシング・モードで，モード=▼111▼，レジスタ=▼001▼とな

り，これらを代入して，

15	14	13	12	11			9	8		6	5			3	2		0	
0	0	1	0	0	0	0	0	0	1	1	1	1	0	0	1			………2079H

となります．

次に，ソースの実効アドレス・オペランドが続き，SRC01のアドレスがロング・ワードでセットされます．ここで，SRC01のアドレスは3000Hから始めて何番地になるのか，SRC01までのすべての命令をハンド・アセンブルして，メモリに占める総バイト数を計算しなくてはなりません．

MOVEA.L命令は６バイト長，**MOVE.W**命令は２バイト長，**CMP.W　D0，D1**命令は２バイト長，**CMPA.L　#$1FFFF，A1**命令は６バイト長，**CMPA.L　A2，A3**命令は２バイト長，**CMPI.B　#$56，(A2)＋**命令は４バイト長，**CMPM．L　(A1)＋，(A2)＋**命令は２バイト長，**TRAP　#13**命令は２バイト長となりますから，総計40バイト長となります．3000Hから始まりますから，SRC01のアドレスは**00003028**H番地となります．

したがって，**MOVEA.L　SRC01，A0**のマシン語は，

207900003028H

となります．

MOVEA.L　SRC02，A1
MOVEA.L　SRC03，A2

の各命令もまったく同様にして，

22790000302CH
247900003030H

がマシン語となります．

MOVE.W　(A0)，D0命令のマシン語は，**MOVE**命令のマシン語フォーマット；

15 14	13 12	11　　9 8　　6	5　　3 2　　0
0　0	サイズ	デスティネーション (レジスタ)　(モード)	ソース (モード)　(レジスタ)

に，サイズ＝ワード＝▼11▼，ソース・フィールドは，モード＝アドレス・レジスタ間接＝▼010▼，レジスタ＝A０＝▼000▼，デスティネーション・フィー

ルドは，モード＝データ・レジスタ直接＝▼000▼，レジスタ＝D0＝▼000▼となり，以上をマシン語フォーマットに代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	………3010H

3010Hが**MOVE.W　(A0)，D0**のマシン語となります．

MOVE.W　(A1)，D1のマシン語は同様にして，

3211H

となります．

CMP.W　D0，D1命令のマシン語は，**CMP**命令のマシン語フォーマット；

15	14	13	12	11　　9	8　　6	5　　0
1	0	1	1	レジスタ	OPモード	実効アドレス

に，レジスタ＝D1＝▼001▼，オペレーション・モード＝ワード・オペレーション＝▼001▼，実効アドレス・フィールド＝データ・レジスタ直接ですから，モード＝▼000▼，レジスタ＝D0＝▼000▼となり，これらを代入して，

15	14	13	12	11		9	8		6	5					0	
1	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	………B240H

B240Hが**CMP.W　D0，D1**のマシン語となります．

CMPA.L　#$1FFFF，A1命令のマシン語は，**CMPA**命令のマシン語フォーマット；

15	14	13	12	11　　9	8　　6	5　　0
1	0	1	1	レジスタ	OPモード	実効アドレス

に，レジスタ＝A11＝▼001▼，オペレーション・モード＝ロング・ワード・オペレーション＝▼111▼，実効アドレス＝即値の＃**$1FFFF**ですから，モード＝▼111▼，レジスタ＝▼100▼となり，これらを代入して，

15	14	13	12	11		9	8		6	5					0	
1	0	1	1	0	0	1	1	1	1	1	1	1	1	0	0	………B3FCH

B3FCHが最初のワードとなり，次に即値0001FFFFHがきますから，最終的に
B3FC0001FFFFH
が**CMPA.L #$1FFFF, A1**のマシン語となります．

CMPA.L A2, A3命令のマシン語は，CMPA命令のマシン語フォーマット；

15	14	13	12	11 9	8 6	5 0
1	0	1	1	レジスタ	OPモード	実効アドレス

に，レジスタ＝A3＝▼011▼，オペレーション・モード＝ロング・ワード・オペレーション＝▼111▼，実効アドレス＝アドレス・レジスタ直接ですから，モード＝▼001▼，レジスタ＝▼010▼となり，これらを代入して，

15	14	13	12	11		9	8		6	5					0	
1	0	1	1	0	1	1	1	1	1	0	0	1	0	1	0	………B7CAH

B7CAHが**CMPA.L A2, A3**のマシン語となります．

CMPI.B #$56, (A2)+命令のマシン語は，**CMPI**命令のマシン語フォーマット；

15	14	13	12	11	10	9	8	7 6	5 0
0	0	0	0	1	1	0	0	サイズ	実効アドレス

に，サイズ＝バイト・オペレーション＝▼00▼，実効アドレス＝A2によるポスト・インクリメント・アドレス・レジスタ間接ですから，モード＝▼011▼，レジスタ＝▼010▼を代入して，

15	14	13	12	11	10	9	8	7	6	5					0	
0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	0	………0C1AH

0C1AHがマシン語の第1ワードとなり，次に即値の**0056H**がきます．以上から，

0C1A0056H

が**CMPI.B　#$56，(A2)＋**のマシン語となります．

CMPM.L　(A1)＋，(A2)＋命令のマシン語は，**CMPM**命令のマシン語フォーマット；

15	14	13	12	11　9	8	7　6	5	4	3	2　0
1	0	1	1	レジスタRx	1	サイズ	0	0	1	レジスタRy

に，レジスタRx＝A2＝▼010▼，サイズ・フィールド＝ロング・ワード・オペレーション＝▼10▼，レジスタRy＝A1＝▼001▼を代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	1	1	0	1	0	1	1	0	0	0	1	0	0	1	………B589H

B589Hが**CMPM.L　(A1)＋，(A2)＋**のマシン語となります．

TRAPのマシン語フォーマットは，

15	14	13	12	11	10	9	8	7	6	5	4	3　0
0	1	0	0	1	1	1	0	0	1	0	0	トラップ・ベクタ番号

で，トラップ・ベクタ番号フィールドに#13を代入して，**TRAP　#13**　のマシン語は，

4E4DH

となります．

以上をまとめると**表3.7**のようなハンド・アセンブル・リストが得られます．

表3.7　例題11のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	207900003028		MOVEA.L	SRC01, A0
003006	22790000302C		MOVEA.L	SRC02, A1
00300C	247900003030		MOVEA.L	SRC03, A2
003012	3010		MOVE.W	(A0), D0
003014	3211		MOVE.W	(A1), D1
003016	B240		CMP.W	D0, D1
003018	B3FC0001FFFF		CMPA.L	#$1FFFF, A1
00301E	B7CA		CMPA.L	A2, A3
003020	0C1A0056		CMPI.B	#$56, (A2)+
003024	B589		CMPM.L	(A1)+, (A2)+
003026	4E4D		TRAP	#13
003028	00010000	SRC01	DC.L	$10000
00302C	00011000	SRC02	DC.L	$11000
003030	00012000	SRC03	DC.L	$12000
			END	

3.9　クリア命令，テスト命令

3.9.1　クリア命令のマシン語

CLR（クリア）命令は，デスティネーションの内容をゼロ・クリアします．

CLR　**<EA>**　………　**0→<EA>**

命令実行後，コンディション・コードは次のようになります．

X	N	Z	V	C
—	0	1	0	0

すなわち，Zフラグがセットされ，Xフラグは変化せず，他はすべてゼロ・クリアされます．

次に，**CLR**命令のマシン語フォーマットを示します．

15	14	13	12	11	10	9	8	7　6	5　　　0
0	1	0	0	0	0	1	0	サイズ	実効アドレス

サイズ・フィールドはオペレーション・サイズを指定するところで,

サイズ・フィールド

0 0………バイト・オペレーション

0 1………ワード・オペレーション

1 0………ロング・ワード・オペレーション

となり，実効アドレス・フィールドでデスティネーションのアドレッシング・モードを指定します.

3.9.2　テスト命令のマシン語

TST（テスト）命令は，デスティネーションの内容をゼロと比較し，その結果をコンディション・コードにセットします．もちろんオペランドの内容は不変です．

TST　〈EA〉 ……… 0－〈EA〉→フラグのみ変化

テストして，すなわちゼロからデスティネーションの内容を減算して，その結果が負ならばNフラグがセットされ，ゼロならばZフラグがセットされ，それ以外のときはすべてリセットされます．8086では**TEST**命令はロジカルANDが行なわれましたが，68000では**CMP**命令の一種として動作します．

TST命令のマシン語フォーマットは,

15	14	13	12	11	10	9	8	7　6	5　　　0
0	1	0	0	1	0	1	0	サイズ	実効アドレス

で，サイズ・フィールドは,

0 0………バイト・オペレーション

0 1………ワード・オペレーション

1 0………ロング・ワード・オペレーション

となり，実効アドレス・フィールドでデスティネーションのアドレッシング・モードを指定します.

3.10 論理演算命令

論理演算命令には論理積(AND)命令，論理和(OR)命令，排他的論理和(EOR)命令，1の補数化(NOT)命令が用意されています．

3.10.1 論理積命令のマシン語

論理積命令にはAND，ANDIが用意されています．

(1) **AND (logical AND) 命令**

AND命令は，ソースとデスティネーションの内容を論理積し，その結果をデスティネーションへ格納します．コンディション・コードは，NとZフラグが変化し，VとCフラグはともにゼロ・クリアされ，Xフラグは前のままで変化しません．

AND　〈EA〉, Dn ……… **Dn・〈EA〉 ⟶Dn**
AND　Dn, 〈EA〉 ……… **〈EA〉・Dn⟶〈EA〉**

X	N	Z	V	C
–	*	*	0	0

オペレーション・サイズは，バイト，ワード，ロング・ワードが可能で，マシン語フォーマットは次のとおりです．

15	14	13	12	11　　9	8　　6	5　　0
1	1	0	0	レジスタ	OPモード	実効アドレス

レジスタ・フィールドは，データ・レジスタDnのNo.を指定し，実効アドレス・フィールドはアドレッシング・モードを指定します．

オペレーション・モード・フィールドは，オペレーションのサイズとDnがソースとして用いられるのか，それともデスティネーションとして用いられるのかを指定します．

オペレーション・モード

バイト	ワード	ロング・ワード	オペレーション
0 0 0	0 0 1	0 1 0	Dn・<EA> → Dn
1 0 0	1 0 1	1 1 0	<EA>・Dn → <EA>

ソースにDnがくる場合，デスティネーションの <EA> のアドレッシング・モードに，アドレス・レジスタ直接とインデックス，ディスプレースメント付プログラム・カウンタ相対を用いることはできません．

デスティネーションにDnがくる場合，ソースの <EA> アドレッシング・モードに，アドレス・レジスタ直接を用いることはできません．つまり，アドレス・レジスタAnとDnとのANDをとることはできないわけです．

(2) **ANDI (AND Immediate) 命令**

ソースの即値と，デスティネーションの内容とを論理積し，その結果をデスティネーションへ格納します．コンディション・コードの変化は，AND命令の場合と同様です．

ANDI #<データ>, <EA> …… <EA>・#<データ> → <EA>

オペレーションサイズは，バイト，ワード，ロング・ワードが可能で，マシン語フォーマットは図3.3に示すとおりです．

また，サイズ・フィールドでオペレーション・サイズの指定を行ない，サイズ・フィールドは次のとおりです．

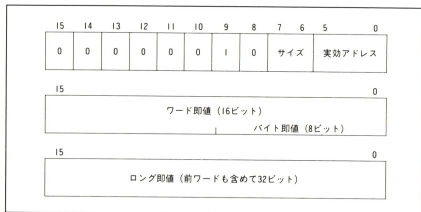



図3.3　ANDI命令のマシン語フォーマット

00……バイト・オペレーション

01……ワード・オペレーション

10……ロング・ワード・オペレーション

実効アドレス・フィールドは，デスティネーションのアドレッシング・モードを指定し，次の即値データ・フィールドに即値データが入ります．

サイズ＝▼00▼のときはバイト・オペレーションで，即値データの8ビット(1バイト）がバイト即値フィールドのところに格納されます．

サイズ＝▼01▼のときはワード・オペレーションで，即値データの1ワード(16ビット）がワード即値フィールドのところに格納され，サイズ＝▼10▼のときはロング・ワード・オペレーションで，即値データのロング・ワード（2ワード）がロング・ワード即値フィールドのところに格納されます．

3.10.2　論理和命令のマシン語

論理和命令には**OR**，**ORI**が用意されています．

(1)　**OR (logical OR)** 命令

OR命令は，ソースとデスティネーションの内容を論理和し，その結果をデスティネーションへ格納します．コンディション・コードは，**AND**命令と同様で，NとZフラグが変化し，VとCフラグがゼロ・クリアされ，Xフラグは前のままで変化しません．

OR　〈EA〉，Dn　……　Dn ∨ 〈EA〉 →Dn

OR　Dn，〈EA〉……　〈EA〉∨ Dn→ 〈EA〉

X	N	Z	V	C
−	*	*	0	0

オペレーション・サイズは，バイト，ワード，ロング・ワードが可能で，マシン語フォーマットは次のとおりです．

15	14	13	12	11　　9	8　　6	5　　0
1	0	0	0	レジスタ	OPモード	実効アドレス

オペレーション・モード・フィールドは，オペレーションのサイズと，デスティネーションがDnか〈EA〉かを指定するのに使用します．

オペレーション・モード

バイト	ワード	ロング・ワード	オペレーション
0 0 0	0 0 1	0 1 0	Dn∨ <EA> → Dn
1 0 0	1 0 1	1 1 0	<EA>∨Dn → <EA>

レジスタ，実効アドレス・フィールドの用い方は，**AND**命令の場合と同様です．

(2) ORI (OR Immediate) 命令

ソースにある即値データと，デスティネーションの内容を論理和し，その結果をデスティネーションへ格納します．コンディション・コードの変化は，**OR**命令と同様です．

ORI #<データ>，<EA> …… <EA>∨#<データ> → <EA>

オペレーション・サイズは，バイト，ワード，ロング・ワードが可能で，マシン語フォーマットは**図3.4**のとおりです．

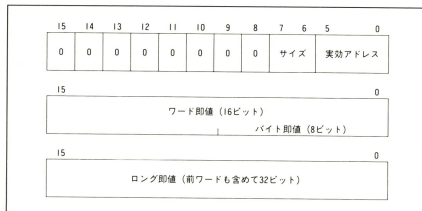



図3．4　ORI命令のマシン語フォーマット

サイズ・フィールドでオペレーション・サイズの指定を行ない，サイズ・フィールドが，

00……バイト・オペレーション

01……ワード・オペレーション

10……ロング・ワード・オペレーション

となるのは，**ANDI**命令と同様です．

また，実効アドレス・フィールド，即値データ・フィールドの指定の仕方も，ANDI命令の場合とまったく同じです．

3.10.3 排他的論理和命令のマシン語

排他的論理和命令には，EOR，EORIが用意されています．

(1) **EOR（Exclusive OR logical）命令**

EOR命令は，ソースとデスティネーションの内容を排他的論理和し，その結果をデスティネーションへ格納します．

EOR命令のソースには，データ・レジスタDnしか指定することができません．コンディション・コード は，他の論理演算命令と同様で，NとZフラグが変化し，VとCフラグがゼロ・クリアされ，Xフラグは前のままで変化しません．

EOR Dn, <EA> ……… **<EA> ⊕ Dn→ <EA>**

X	N	Z	V	C
−	*	*	0	0

オペレーション・サイズは，バイト，ワード，ロング・ワードが可能で，マシン語フォーマットは次のようになります．

15	14	13	12	11　　9	8　　6	5　　0
1	0	1	1	レジスタ	OPモード	実効アドレス

レジスタ・フィールドは，ソースで用いられるデータ・レジスタDnのNo.を指定するフィールドで，オペレーション・モード・フィールドは，オペレーションのサイズを指定するのに用いられます．

オペレーション・モード

バイト	ワード	ロング・ワード	オペレーション
1 0 0	1 0 1	1 1 0	<EA> ⊕ Dn→ <EA>

実効アドレス・フィールドで，デスティネーション<EA>のアドレッシング・モードの指定を行ないます．

(2) EORI（Exclusive OR Immediate）命令

ソースにある即値データと，デスティネーションの内容を排他的論理和し，その結果をデスティネーションへ格納します．コンディション・コードの変化は，EOR命令と同様です．

EORI　#〈データ〉，〈EA〉……〈EA〉⊕#〈データ〉→〈EA〉

オペレーション・サイズは，バイト，ワード，ロング・ワードが可能です．図3.5に，マシン語フォーマットを示します．

第１バイト目を除いて，ORI命令と同様のフォーマットとなります．

図３．５　EORI命令のマシン語フォーマット

サイズ・フィールドでオペレーション・サイズの指定を行ないますが，これは，

サイズ・フィールド
- 00……バイト・オペレーション
- 01……ワード・オペレーション
- 10……ロング・ワード・オペレーション

となり，これもANDI命令と同様です．実効アドレス・フィールド，即値データ・フィールドの指定の方法も，ANDI命令のときと同様になります．

3.10.4　NOT命令のマシン語

1の補数化命令として，NOT（Logical Complement）命令が用意されています．

このNOT命令は，デスティネーションの内容の1の補数をとり，これをデスティネーションに格納します．1の補数とは，すべてのビットの値を反転してやれば作られます．コンディション・コードの変化は，他の論理演算命令と同様で，N，Zは変化し，V，Cはゼロクリアされ，Xは前のままの値で変化しません．

次に，NOT命令のマシン語フォーマットを示します．

15	14	13	12	11	10	9	8	7 6	5 0
0	1	0	0	0	1	1	0	サイズ	実効アドレス

サイズ・フィールドでオペレーション・サイズを指定し，他の命令と同様に，

サイズ・フィールド

00……バイト・オペレーション

01……ワード・オペレーション

10……ロング・ワード・オペレーション

となり，また実効アドレス・フィールドで，デスティネーション・オペランドのアドレッシング・モードを指定します．

それでは次に，論理演算命令のマシン語変換の例題を見てみましょう．

3.11　論理演算命令のマシン語プログラミング例

例題12　次の論理演算命令のプログラムをマシン語に変換しなさい(ハンド・アセンブルせよ).

```
          ORG       $3000
          EOR.L     D0, D0
          AND.L     D0, D1
          AND.B     D1, MASK1
          AND.L     MASK2, D2
          ANDI.L    #$0FF, D2
          ANDI.B    #$7F, D1
          ORI.B     #$80, D2
          NOT.W     5(A1)
          OR.W      D1, (A1)+
          TRAP      #13
MASK1     DC.B      $0
MASK2     DC.L      $0FF
          END
```

解き方

EOR.L　D0, D0命令は，排他的論理和命令**EOR**のマシン語フォーマットを用いて，マシン語を作成します．

15	14	13	12	11　9	8　6	5　0
1	0	1	1	レジスタ	OPモード	実効アドレス

レジスタ・フィールドは，ソース・オペランドのデータ・レジスタDnのNo.を指定するところで，ソースはD0ですから，レジスタ・フィールド＝▼000▼となります．

オペレーション・モード・フィールドは，オペレーション・サイズを指定

するところで，ロング・オペレーションがここでは行なわれますから，オペレーション・モード・フィールド＝ロング・ワード・オペレーション＝▼110▼となります．

実効アドレス・フィールドは，デスティネーション〈EA〉のアドレッシング・モードを指定するところで，デスティネーション・オペランドはD0ですから，D0レジスタを用いたデータ・レジスタ直接アドレッシング・モードとなり，48ページの表3．3より，▼000000▼のビット・パターンが実効アドレス・フィールドにセットされます(ここで，もう一度，表3．3を下に掲載しておきます)．これらを代入して，

15	14	13	12	11　9	8　6	5　0	
1	0	1	1	0 0 0	1 1 0	0 0 0 0 0 0	………B180H

B180Hが**EOR.L　D0，D0**のマシン語となります．

AND.L　D0，D1命令は，**AND**命令のマシン語フォーマット；

15	14	13	12	11　9	8　6	5　0
1	1	0	0	レジスタ	OPモード	実効アドレス

表3．3　モード，レジスタ・フィールドによって決められるアドレッシング・モード

モード・フィールド	レジスタ・フィールド	アドレッシング・モード
0 0 0	Dn	データ・レジスタ直接
0 0 1	An	アドレス・レジスタ直接
0 1 0	An	アドレス・レジスタ間接
0 1 1	An	ポスト・インクリメント・アドレス・レジスタ間接
1 0 0	An	プリ・デクリメント・アドレス・レジスタ間接
1 0 1	An	ディスプレースメント付アドレス・レジスタ間接
1 1 0	An	インデックス，ディスプレースメント付アドレス・レジスタ間接
1 1 1	0 0 0	アブソリュート・ショート
1 1 1	0 0 1	アブソリュート・ロング
1 1 1	0 1 0	ディスプレースメント付プログラム・カウンタ相対
1 1 1	0 1 1	インデックス，ディスプレースメント付プログラム・カウンタ相対
1 1 1	1 0 0	即　値

において，レジスタ・フィールド＝D1＝▼001▼，オペレーション・モード・フィールド＝▼010▼，実効アドレス・フィールド＝▼000000▼を代入して，

15	14	13	12	11　9	8　6	5　0	
1	1	0	0	0 0 1	0 1 0	0 0 0 0 0 0	………C280H

C280HがAND.L　D0，D1のマシン語となります．

AND.B　D1，MASK1 命令は，上と同じマシン語フォーマットにおいて，レジスタ・フィールド＝D1＝▼001▼，オペレーション・モード・フィールド＝▼100▼，実効アドレス・フィールド＝アブソリュート・ロング＝▼111001▼を代入して，

15	14	13	12	11　9	8　6	5　0	
1	1	0	0	0 0 1	1 0 0	1 1 1 0 0 1	………C339H

C339Hがマシン語の第1ワードとなります．

次に，デスティネーションの実効アドレス・オペランドが続き，**MASK1**のアドレスがロング・ワードでセットされます．3000Hから始めて**MASK1**のアドレスは，00003026H番地となりますから，**AND.B　D1，MASK1**のマシン語は，

C33900003026H

となります．

AND.L　MASK2，D2 命令も前命令と同じマシン語フォーマットにおいて，レジスタ・フィールド＝D2＝▼010▼，オペレーション・モード・フィールド＝▼010▼，実効アドレス・フィールド＝アブソリュート・ロング＝▼111001▼を代入して，

15	14	13	12	11　9	8　6	5　0	
1	1	0	0	0 1 0	0 1 0	1 1 1 0 0 1	………C4B9H

C4B9Hがマシン語の第1ワードとなります．次に，ソースの実効アドレス・オペランドが続き，**MASK2**のアドレスがロング・ワードでセットされます．

MASK2のアドレスは，00003028H番地となりますから，**AND.L　MASK2，D2**のマシン語は，

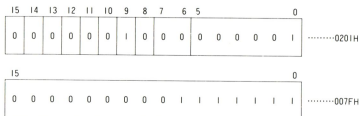
C4B900003028H

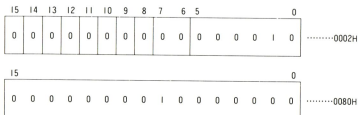
となります．

ANDI.L　#$0FF，D2命令は，ANDI命令のマシン語フォーマット，

において，サイズ・フィールド＝ロング・ワード＝▼10▼，実効アドレス・フィールド＝デスティネーションのアドレッシング・モード＝データ・レジスタ直接(D2)＝▼000010▼，即値データ・フィールド＝ロング即値＝▼000000FF▼を代入して，

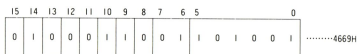
0282000000FFHが，**ANDI.L　#$0FF，D2**のマシン語となります．

ANDI.B　#$7F，D1のマシン語は，前命令と同じマシン語フォーマットに，サイズ＝バイト＝▼00▼，実効アドレス＝データ・レジスタ直接(D1)＝▼000001▼，即値データ・フィールド＝バイト即値＝▼007FH▼を代入して，

0201007FHが，ANDI.B　#$7F，D1のマシン語となります．

ORI.B　#$80，D2命令は，ORIのマシン語フォーマットに，サイズ＝バイト・オペレーション＝"00"，実効アドレス＝データ・レジスタ(D2)直接＝"000010"，即値フィールド＝バイト即値＝"0080H"を代入して，

00020080Hが，ORI.B　#$80，D2のマシン語となります．

NOT.W　5(A1)　命令は，NOTのマシン語フォーマット；

において，サイズ＝ワード・オペレーション＝"01"，実効アドレス＝ディスプレースメント付アドレス・レジスタ間接＝"101001"を代入して，

15	14	13	12	11	10	9	8	7	6	5					0	
0	1	0	0	0	1	1	0	0	1	1	0	1	0	0	1	………4669H

4669Hがマシン語の第１ワードとなります．ディスプレースメントは5ですから，これを次に持ってきて，**NOT.W　5(A1)** のマシン語は，

46690005H

となります．

OR.W　D1, (A1)＋命令は，**OR**命令のマシン語フォーマットに，レジスタ・フィールド＝▼101▼，オペレーション・モード・フィールド＝▼101▼，実効アドレス・フィールド＝ポスト・インクリメント・アドレス・レジスタ（A1）間接＝▼011001▼を代入して，

15	14	13	12	11		9	8		6	5					0	
1	0	0	0	0	0	1	1	0	1	0	1	1	0	0	1	………8359H

8359Hが**OR.W　D1,　(A1)＋**のマシン語となります．

TRAPのマシン語フォーマットは，

15	14	13	12	11	10	9	8	7	6	5	4	3　　0
0	1	0	0	1	1	1	0	0	1	0	0	トラップ・ベクタ番号

で，トラップ・ベクタ番号フィールドに#13を代入して，**TRAP　#13**　のマシン語は，

4E4DH

となります．

以上をまとめると，次ページの**表３．８**に示すようなハンド・アセンブル・リストが得られます．

表3.8 例題12のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	B180		EOR.L	D0, D0
003002	C280		AND.L	D0, D1
003004	C33900003026		AND.B	D1, MASK1
00300A	C4B900003028		AND.L	MASK2, D2
003010	0282000000FF		ANDI.L	#$0FF, D2
003016	0201007F		ANDI.B	#$7F, D1
00301A	00020080		ORI.B	#$80, D2
00301E	46690005		NOT.W	5(A1)
003022	8359		OR.W	D1, (A1)+
003024	4E4D		TRAP	#13
003026	00	MASK1	DC.B	$0
003028	000000FF	MASK2	DC.L	$0FF
			END	

3.12 テスト・アンド・セット命令

マルチ・プロセッサ・システムで共有メモリをアクセスする際に用いられる命令に，テスト・アンド・セット命令があります．このテスト・アンド・セット命令のマシン語変換を行なう前に，セマフォーオペレーションについて，簡単に見てみましょう．

3.12.1 セマフォーオペレーション

マルチ・プロセッサ・システムで問題になるのは，共有メモリへのアクセスの制御があります．すなわち，**図3.6**に示すように，CPU#1とCPU#2とがあって，これらはともに共有メモリをアクセスすることができる場合，片方のCPUが共有メモリ中の共有データを更新している最中に，他方のCPUがそのデータを読み取ってしまうことも考えられます．更新が完全に終了していない誤ったデータをリードしてしまうことになり，正しいデータが2つのCPU間で受け渡すことができなくなってしまいます．

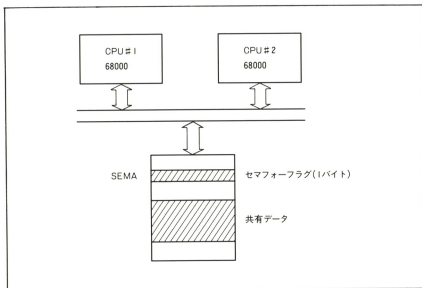



図3.6　68000マルチ・プロセッサ・システム

これを解決するには，共有メモリ中に▼共有メモリ使用中▼のフラグを設け，フラグが▼1▼の間は，他のCPUは共有メモリへのアクセスを起こさない，またフラグが▼0▼のときは，共有メモリが使用されていないわけですから，共有メモリへのアクセスは許されるというようにすればよいことがわかります．

この共有メモリ中に設けられたフラグのことを，セマフォーといい，このセマフォーを用いてマルチCPUの共有メモリへのアクセス制御を行なうことを，セマフォーオペレーションと呼んでいます．共有メモリへアクセスする場合，どのCPUも**図3.7**(次ページ)のような手順を経て，アクセスしなくてはなりません．

このセマフォーオペレーションを実現するために，68000に用意されている命令が“テスト・アンド・セット（**TAS**）命令”で，セマフォーオペレーションはこの命令を使ってプログラムされます．おのおののCPUのプログラムは，共有メモリをアクセスする際には，必ずこのセマフォーオペレーションをするように，プログラムを作っておかなくてはなりません．

すなわち，これがどちらか一方のCPUのプログラムが正しくコーディングされていないで，誤っていた場合，セマフォーオペレーションは保障されなくな

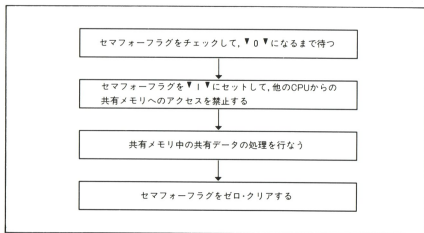



図3．7　セマフォーオペレーション

ります．たとえば，共有メモリに対する処理が終了したら，セマフォーフラグを必ずゼロ・クリアしておかなくてはいけません．これを忘れると，フラグは常に▼1▼になったままで，永久にどのCPUも共有メモリにアクセスすることができなくなってしまいます．

こうしたことが生じないように，すべてのCPUのプログラムは，正しくセマフォーオペレーションを実行するように記述されていなくてはなりません．

3.12.2　テスト・アンド・セット（TAS）命令のマシン語

TAS（Test And Set operand）命令はバイトのオペランドをテストして，その結果，コンディション・コードのNフラグとZフラグが影響されます．さらに，同一バス・サイクル中にバイト・オペランドの最上位ビットを▼1▼にセットします．

X	N	Z	V	C
—	*	*	0	0

オペレーション・サイズはバイトのみが可能で，マシン語フォーマットは，次のとおりです．

15	14	13	12	11	10	9	8	7	6	5　　　　0
0	1	0	0	1	0	1	0	1	1	実効アドレス

実効アドレス・フィールドは，オペランドのアドレッシング・モードを指定します．次に，TAS命令のマシン語変換の例を考えてみましょう．

3.13 テスト・アンド・セット(TAS)命令のマシン語プログラミング例

例題13

次のセマフォーオペレーションのプログラムをマシン語に変換しなさい(ハンド・アセンブルせよ)．

```
        ORG      $3000
AGAIN   TAS      SEMA
        BMI.S    AGAIN
        MOVE.W   D0,TIME1
        CLR.B    SEMA
        TRAP     #13
        ORG      $10000
SEMA    DC.B     0
TIME1   DC.W     0
        END
```

解き方

TAS命令のマシン語フォーマットは，

15	14	13	12	11	10	9	8	7	6	5 ～ 0
0	1	0	0	1	0	1	0	1	1	実効アドレス

で，実効アドレス・フィールドでオペランドのアドレッシング・モードを指定します．▼アブソリュート・ロング▼のアドレッシング・モードですから，▼111001▼を実効アドレス・フィールドに代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	0	0	1	0	1	0	1	1	1	1	1	0	0	1	………4AF9H

4AF9Hがマシン語の第1ワードとなります．次に，オペランドの実効アドレスが続き，SEMAのアドレスがロング・ワードでセットされます．SEMEの番地は，00010000H番地ですから，**TAS　SEMA**のマシン語は，

4AF900010000H

となります．

BMI.S　AGAINのマシン語フォーマットは，

で，条件フィールドはMI＝▼1011▼，DISP8＝▼F8▼を代入して，

6BF8Hが**BMI.S　AGAIN**のマシン語となります．

MOVE.W　D0, TIME1はセマフォーチェックした後，共有メモリ中の共有データTIME1を変更する命令で，**MOVE**命令のマシン語フォーマットは，

15	14	13　12	11　　9　8　　6		5　　3　2　　0	
0	0	サイズ	デスティネーション （レジスタ）	（モード）	ソース （モード）	（レジスタ）

となります．

サイズ・フィールドはワード・オペレーションですから，▼11▼をセットします．ソース・オペランドはD0で，アドレッシング・モードはデータ・レジスタ直接，したがって，モード・フィールド＝▼000▼，レジスタ・フィールド＝▼000▼となります．

デスティネーションはTIME1で，アブソリュート（絶対）ロングのアドレッシング・モードですから，モード・フィールド＝▼111▼，レジスタ・フィールド＝▼001▼となり，以上をMOVE命令のマシン語フォーマットに代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	………33C0H

33C0Hがマシン語の第1ワードとなります．これに続けて，デスティネーション実効アドレスのオペランド・ワードがきますが，TIME1のアドレスは00010002Hですから以上をまとめて，

33C000010002H

となり，これが**MOVE.W　D0，TIME1**のマシン語となります．

CLR命令のマシン語フォーマットは，

15	14	13	12	11	10	9	8	7　6	5　　　0
0	1	0	0	0	0	1	0	サイズ	実効アドレス

で，サイズ・フィールドでオペレーションのサイズを決定し，クリア命令はすべてのサイズ指定が可能となっています．

００………バイト・オペレーション
０１………ワード・オペレーション
１０………ロング・ワード・オペレーション

セマフォーフラグSEMAはバイトで，**CLR.B　SEMA**命令はバイト・オペレーション，したがってサイズ・フィールド＝▼00▼となります．

実効アドレス・フィールドはアブソリュート・ロングで，ビット・パターンは▼111001▼となり，以上を代入して，

4239Hがマシン語の第１ワード目となり，次にSEMAの番地がロング・ワードで続き，このアドレスは，00010000H番地となりますから，以上より**CLR SEMA**のマシン語は，

423900010000H

となります．

TRAPのマシン語フォーマットは，

15	14	13	12	11	10	9	8	7	6	5	4	3〜0
0	1	0	0	1	1	1	0	0	1	0	0	トラップ・ベクタ番号

で，トラップ・ベクタ番号フィールドに#13を代入して，**TRAP　#13**のマシン語は，

4E4DH

となります．

以上をまとめると，**表3.9**のようなハンド・アセンブル・リストができます．

表3．9　例題13のハンド・アセンブル・リスト

```
       00003000                  ORG      $3000
003000 4AF900010000    AGAIN     TAS      SEMA
003006 6BF8                      BMI.S    AGAIN
003008 33C000010002              MOVE.W   D0, TIME1
00300E 423900010000              CLR.B    SEMA
003014 4E4D                      TRAP     #13
       00010000                  ORG      $10000
010000 00              SEMA      DC.B     0
010002 0000            TIME1     DC.W     0
                                 END
```

3.14　BCD演算命令

BCD（Binary Coded Decimal; 2進化10進数）用の演算命令として，加算，減算，補数化の命令が68000には用意されており，これらの命令を用いて，**BCD**演算を行なうことができます．**BCD**の加算命令は**ABCD**（Add Binary Coded Decimal ; Add BCD），**BCD**の減算命令は**SBCD**（Subtract Binary Coded Decimal ; Subtract BCD），**BCD**の補数化命令は**NBCD**(Negate Binary Coded Decimal ; Negate BCD)のニーモニックが使われます．

3.14.1　ABCD命令のマシン語

ABCD（Add BCD）命令は，**BCD**のオペランド（8ビット長，1バイト）間で**BCD**加算を行ない，これにさらにXフラグの内容を加算し，その結果をデスティネーション・オペランドに格納します．オペランドには，データ・レジスタ間，メモリ間の2とおりの指定ができます．すなわち，

①　データ・レジスタとデータ・レジスタの**BCD**加算

　　ABCD　Dy，Dx

②　メモリとメモリの**BCD**加算

　　ABCD　－(Ay)，－(Ax)

②のメモリとメモリの**BCD**加算で用いられるメモリ・アドレッシング・モー

ドは"プリ・デクリメント・アドレス・レジスタ間接"が必ず用いられ，これ以外のモードを使うことは許されません．

ABCD命令はバイト・オペレーションのみが可能で，他のオペレーションはできません．

ABCD　Dy，Dx……………………**Dx＋Dy＋Xフラグ→Dx**

ABCD　－(Ay)，－(Ax)……　**(Ax)＋(Ay)＋フラグ→(Ax)**

X	N	Z	V	C
*	U	*	U	*

コンディション・コードはNとVフラグが不定で，10進演算でのキャリーが発生すればC，Xにセットされ，Zフラグは，結果が0でないならゼロ・クリアされ，その他の場合は変化しません．

オペレーション・サイズは，先ほど述べたようにバイトのみが可能となります．マシン語フォーマットは，次のとおりです．

15	14	13	12	11～9	8	7	6	5	4	3	2～0
1	1	0	0	レジスタRx	1	0	0	0	0	R/M	レジスタRy

レジスタRxフィールドは，デスティネーション・オペランドのレジスタNo.を指定するところで，R／M＝0のときはデータ・レジスタのNo.を，またR／M＝1のときはプリ・デクリメント・アドレス・レジスタ間接で用いるアドレス・レジスタのNo.を指定します．

R／Mフィールドは，オペランドのアドレッシング・モードを指定する1ビットからなるフィールドで，

R／M＝0のとき……データ・レジスタ間オペレーション

R／M＝1のとき……メモリ間オペレーション

が指定されます．

レジスタRyフィールドは，ソース・オペランドのレジスタNo.を指定するところで，Rxフィールドの場合と同様に，R／M＝0のときはデータ・レジスタのNo.を指定し，R／M＝1のときはプリ・デクリメント・アドレス・レジスタ間接で用いるアドレス・レジスタのNo.を指定します．

3.14.2　SBCD命令のマシン語

SBCD (Subtract BCD) 命令は，デスティネーション・オペランドからソース・オペランドとXフラグの内容をBCDで減算し，その結果をデスティネーション・オペランドへ格納します．オペランドには，データ・レジスタ間，またはメモリ間の2とおりの指定ができます．

①　データ・レジスタとデータ・レジスタとのBCD減算

　　SBCD　Dy, Dx

②　メモリとメモリのBCD減算

　　SBCD　－(Ay)，－(Ax)

②のメモリ間でのBCD減算で用いられるメモリ・アドレッシング・モードは，“プリ・デクリメント・アドレス・レジスタ間接”で，他のモードを用いることはできません．

SBCD命令はバイト・オペレーションのみが可能で，1バイトに入った2個のBCD数値に対してBCD減算が行なわれます．

　　SBCD　Dy, Dx ……………………**Dx－Dy－Xフラグ→Dx**

　　SBCD　－(Ay)，－(Ax) ……　**(Ax)－　(Ay)－Xフラグ→ (Ax)**

X	N	Z	V	C
*	U	*	U	*

コンディション・コードの変化は，ABCD命令の場合と同様で，NとVフラグが不定で，キャリーが発生すればC，Xにセットされ，Zフラグは結果が0でなければゼロ・クリアされ，それ以外は変化しません．

マシン語フォーマットは，次のとおりです．

15	14	13	12	11　　9	8	7	6	5	4	3	2　　0
1	0	0	0	レジスタRx	1	0	0	0	0	R/M	レジスタRy

レジスタRxフィールドは，デスティネーション・オペランドのレジスタNo.を指定し，R／M＝0のときはデータ・レジスタのNo.を，またR／M＝1のときはプリ・デクリメント・アドレス・レジスタ間接で用いるアドレス・レジスタのNo.を指定します．

R／Mフィールドはオペランドのアドレッシング・モードを指定する1ビット

のフィールドで，
　　R／M＝０のとき……データ・レジスタ間オペレーション
　　R／M＝１のとき……メモリ間オペレーション
が指定されます．
　レジスタRyフィールドは，ソース・オペランドのレジスタNo.を指定するところで，Rxフィールドの場合と同様に，R／M＝０のときはデータ・レジスタのNo.を指定し，R／M＝１のときはプリ・デクリメント・アドレス・レジスタ間接のアドレッシング・モードで用いるアドレス・レジスタのNo.を指定します．
　以上のフィールドの指定の仕方は**ABCD**命令の場合とまったく同様です．

3.14.3　NBCD命令のマシン語

NBCD（Negate　BCD）命令は，ゼロ（０）からデスティネーション・オペランドの内容と，Xフラグの内容を減算し，この結果をデスティネーション・オペランドに格納します．このオペレーションは10進演算で行なわれますから，Xフラグがゼロのときは10の補数を，Xフラグが１のときは９の補数を作ることになります．
　オペレーションはバイト・オペレーションのみが許されます．

NBCD　〈EA〉………０－**〈EA〉－Xフラグ→〈EA〉**

X	N	Z	V	C
*	U	*	U	*

　コンディション・コードは他の**BCD**命令，**ABCD**，**SBCD**命令の場合とまったく同様です．
　マシン語は次のとおりです．

15	14	13	12	11	10	9	8	7	6	5　　　　0
0	1	0	0	1	0	0	0	0	0	実効アドレス

　実効アドレス・フィールドで，デスティネーション・オペランドのアドレッシング・モードの指定を行ないます．
　次に，**BCD**演算命令のマシン語変換の例題を解いてみましょう．

3.15　BCD演算命令のマシン語プログラミング例

例題14　次のプログラムのマシン語を作りなさい.

```
        ORG     $3000
        MOVE    #$04, CCR
LABELA  ABCD    -(A0), -(A1)
        DBRA    D0, LABELA
        MOVE    #$04, CCR
LABELS  SBCD    -(A2), -(A3)
        DBRA    D1, LABELS
        TRAP    #13
        END
```

MOVE　#$04,CCRのマシン語は，**CCR**への転送命令のマシン語フォーマット；

15	14	13	12	11	10	9	8	7	6	5　　　　0
0	1	0	0	0	1	0	0	1	1	実効アドレス

において，実効アドレス・フィールドに即値アドレッシング・モードのビット・パターン▼111100▼を代入して，

15	14	13	12	11	10	9	8	7	6	5					0	
0	1	0	0	0	1	0	0	1	1	1	1	1	1	0	0	………44FCH

となり，この後に即値をもってきてやればよいわけですから，

44FC0004H

となります．

LABELA　ABCD －(A0)，－(A1)のマシン語フォーマットは，前述したように，

15	14	13	12	11　　　　9	8	7	6	5	4	3	2　　　0
1	1	0	0	レジスタRx	1	0	0	0	0	R/M	レジスタRy

となります．

R／Mフィールドは，オペランドのアドレッシング・モードを指定するところで，メモリ間での演算ですから，R／M＝1にセットします．レジスタRxフィールドは，デスティネーションのアドレス・レジスタのNo.をセットするところで，A1の1をセットします．レジスタRyフィールドは，ソース・オペランドのアドレス・レジスタNo.をセットするフィールドで，A0の0をセットします．

以上から，

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	………C308

C308Hが **LABELA　ABCD －(A0)，－(A1)** のマシン語となります．

DBRA　D0，LABELAのマシン語フォーマットは，

で，条件フィールド＝▼0001▼，レジスタ＝D0＝▼000▼，DISP16＝**LABELA**までのディスプレースメント＝FFFCHを代入して，

51C8FFFCHがDBRA　D0,LABELAのマシン語となります．

LABELS　SBCD　−(A2),−(A3)　のマシン語フォーマットは，

15	14	13	12	11–9	8	7	6	5	4	3	2–0
1	0	0	0	レジスタRx	1	0	0	0	0	R/M	レジスタRy

で，これにRx＝デスティネーションのアドレス・レジスタNo.＝3＝▼011▼，Ry＝ソースのアドレス・レジスタNo.＝2＝▼010▼，R／M＝メモリ間オペレーション＝1，これらを代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	………870AH

870AHがマシン語となります．

DBRA　D1,LABELS　のマシン語は，51C9FFFCHとなります．TRAP　#13も4E4DHがマシン語となり，以上をまとめると，次ページの表3.10のようなリストが得られます．

表3.10 例題14のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	44FC0004		MOVE	#$04, CCR
003004	C308	LABELA	ABCD	−(A0), −(A1)
003006	51C8FFFC		DBRA	D0, LABELA
00300A	44FC0004		MOVE	#$04, CCR
00300E	870A	LABELS	SBCD	−(A2), −(A3)
003010	51C9FFFC		DBRA	D1, LABELS
003014	4E4D		TRAP	#13
			END	

3.16 シフト命令

シフト命令には，LSL, LSR, ASL, ASR命令があります．

3.16.1 LSL命令のマシン語

LSL (Logical Shift Left) 命令は，**図3.8**のような**左へ論理シフト**を行ないます．そして，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容を左へ論理シフトし，最上位ビットはC(キャリーフラグ)，X(拡張フラグ)に入り，左へシフトして空になった下位のビットには，0が入ります．

データ・レジスタ内容を論理シフトする場合に，シフトするビット数は，即値（1～8の範囲）で指定することも，またデータ・レジスタを用いて指定することもできます（データ・レジスタを用いる場合，下位6ビット（0～63）が有効）．

図示してあるように，データ・レジスタの左への論理シフトは32ビット，16ビット，8ビットに対して実行可能です．すなわち，バイト，ワード，ロング・ワードのオペレーションが可能なのに対し，メモリ内容の場合は，ワード・オペレーションのみでバイト，ロング・ワードを指定することはできません．さらに，メモリでシフトできるビット数は，常に1ビットで，複数ビットのシフトはできません．

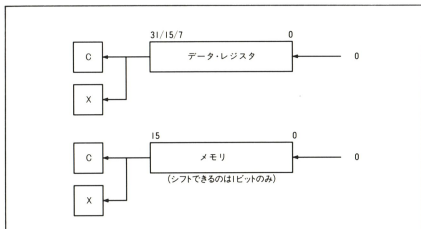



図3.8　LSL命令

LSL　#<即値>, Dy……Dyの内容を#<即値>のビット数だけ左へ論理シフトする．

LSL　Dx,Dy……Dyの内容をDxの内容のビット数だけ左へ論理シフトする．

LSL　<EA>……メモリ <EA> の内容を1ビットだけ左へ論理シフトする．

X	N	Z	V	C
*	*	*	0	*

コンディション・コードは，CとXに最上位ビット（MSB）がセットされるのは，前に述べたとおりです．しかし，シフトするビット数が0のときはCはゼロ・クリアされ，Xは変化しないで，前のままの状態を保ちます．Vフラグは常に0で，NとZはおのおの負，ゼロでセットされ，それ以外のときはゼロ・クリアされます．

マシン語フォーマットは，データ・レジスタ内容をシフトする場合と，メモリ内容をシフトする場合と2とおり用意されています．

■レジスタ内容をシフトする場合：

15	14	13	12	11〜9	8	7〜6	5	4	3	2〜0
1	1	1	0	カウント/レジスタ	1	サイズ	i/r	0	1	レジスタ

カウント/レジスタ・フィールドは，i／r＝０のときはシフトするカウント数（ビット数）が即値で，このフィールドにセットされます．０は8を，また１〜７は１〜７を表わします．i／r＝１のときは，シフトするビット数はデータ・レジスタ内にあり，そのデータ・レジスタのNo.がこのカウント/レジスタ・フィールドにセットされます．

サイズ・フィールドは，オペレーションのサイズを指定するところで，

０ ０……バイト・オペレーション

０ １……ワード・オペレーション

１ ０……ロング・ワード・オペレーション

となります．

i／rフィールドは，シフトするビット数の指定を即値データを用いるか，あるいはデータ・レジスタを用いるかを指定する１ビットのフィールドで，

i／r＝０で即値データ

i／r＝１でデータ・レジスタ

が指定されます．

レジスタ・フィールドは，シフトされるデータの格納されているデータ・レジスタのNo.を指定します．

■メモリ内容をシフトする場合：

15	14	13	12	11	10	9	8	7	6	5〜0
1	1	1	0	0	0	1	1	1	1	実効アドレス

実効アドレス・フィールドは，シフトするメモリのアドレッシング・モードを指定します．

3.16.2　LSR命令のマシン語

LSR（Logical Shift Right）命令は，図3.9のようにオペランド内容を右へ

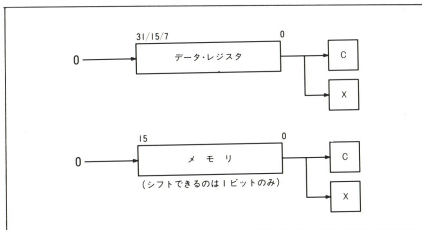



図3.9　LSR命令

論理シフトします．そして，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容を右へ論理シフトし，最下位ビット（LSB）はC（キャリーフラグ），X（拡張フラグ）に入り，右へシフトして空になった上位のビットには0が入ります．

メモリ・オペランドの場合，シフトできるビット数は1ビットで，ワード・オペレーションのみが可能なのは**LSL**命令の場合とまったく同様です．

データ・レジスタがオペランドの場合，バイト，ワード，ロング・ワードと，すべてのオペレーションが可能で，シフトするビット数も**LSL**命令と同様に，即値（1～8）で指定することも，データ・レジスタを用いて指定することもできます（有効ビット数は**LSL**命令の場合と同様）．

LSR　#〈即値〉，Dy　…Dxの内容を#〈即値〉のビット数だけ右へ論理シフトする．

LSR　Dx,Dy……Dyの内容をDxの内容のビット数だけ右へ論理シフトする．

LSR 〈EA〉……メモリ〈EA〉の内容を1ビットだけ右へ論理シフトする．

コンディション・コードの変化は，CとXに最下位ビット（LSB）がセットされる点を除いて，**LSL**の場合と同様です．マシン語フォーマットは，データ・レジスタ内容をシフトする場合と，メモリ内容をシフトする場合と2とおり用意されており，おのおの次のとおりです．

■レジスタ内容をシフトする場合：

15	14	13	12	11〜9	8	7〜6	5	4	3	2〜0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	0	1	レジスタ

カウント/レジスタ・フィールド，サイズ・フィールド，i/rフィールド，レジスタ・フィールドの指定の仕方は，LSL命令のときとまったく同じです．

■メモリ内容をシフトする場合：

15	14	13	12	11	10	9	8	7	6	5〜0
1	1	1	0	0	0	1	0	1	1	実効アドレス

実効アドレス・フィールドもLSL命令の場合と同じ指定の仕方です．

3.16.3　ASL命令のマシン語

ASL（Arithmetic Shift Left）命令は，図3.10のように左へ算術シフトを実行します．そして，デスティネーション・オペランドで指定したデータ・レ

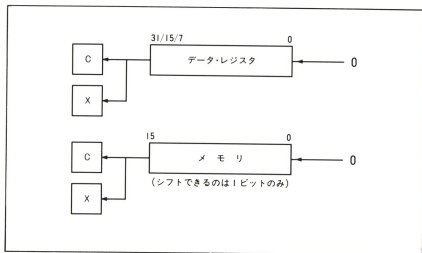



図3.10　ASL命令

ジスタ，またはメモリの内容を左へ算術シフトし，最上位ビット（MSB）はC（キャリーフラグ），X（拡張フラグ）に入り，左へシフトして空になった下位のビットには0が入ります．

論理シフト命令のときと同様に，データ・レジスタ内容を算術シフトする場合は，シフトするビット数を即値（1～8）で，またデータ・レジスタを用いて指定することができます．

データ・レジスタの左への算術シフトは，バイト，ワード，ロング・ワード・オペレーションが可能ですが，メモリの場合は，ワード・オペレーションしかできません．しかも，メモリでシフトできるビット数は1ビットのみです．

ASL　#〈即値〉，Dy……Dyの内容を#〈即値〉のビット数だけ左へ算術シフトする．

ASL　Dx,Dy……Dyの内容をDxの内容のビット数だけ左へ算術シフトする．

ASL 〈EA〉……メモリ〈EA〉の内容を1ビットだけ左へ算術シフトする．

X	N	Z	V	C
*	*	*	*	*

コンディション・コードは，CとXに最上位ビット（MSB）がセットされますが，シフトするビット数（シフト回数）が0のときは，Cはゼロ・クリアされ，Xは変化しないで，前のままの状態を保持します．

Vフラグは，シフト中に一度でも符号が変化すると，1にセットされ，それ以外の場合はゼロ・クリアされます．NとZフラグはおのおの結果が負，ゼロになるとセットされ，それ以外ではゼロ・クリアされます．

マシン語フォーマットは，次のとおりです．

■レジスタ内容をシフトする場合：

15	14	13	12	11　　9	8	7　6	5	4	3	2　　0
1	1	1	0	カウント/レジスタ	1	サイズ	i/r	0	0	レジスタ

（各フィールドの指定の仕方は**LSL**命令とまったく同様です）．

■メモリ内容をシフトする場合：

15	14	13	12	11	10	9	8	7	6	5　　　　　　　0
1	1	1	0	0	0	0	1	1	1	実効アドレス

（実効アドレス・フィールドは，メモリのアドレッシング・モードを指定）．

3.16.4 ASR命令のマシン語

ASR（Arithmetic Shift Right）命令は，**図3.11**のようにオペランドの内容を**右へ算術シフト**します．そして，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容を右へ算術シフトし，最下位ビット（LSB）はC（キャリーフラグ），X（拡張フラグ）に入り，右へシフトして空になった上位のビットには，サイン・ビットがそのままシフトされて入り，サイン・ビットはそのままの値が残ります．

メモリ・オペランドの場合，シフトできるビット数は1で，ワード・オペレーションだけが許されるのも**LSR**命令の場合と同じです．データ・レジスタがオペランドの場合，バイト，ワード，ロング・ワードの全オペレーションが可

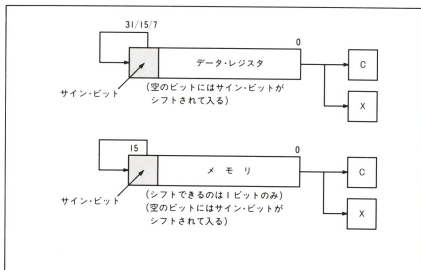



図3.11 ASR命令

能で，シフトのビット数も即値（1～8）とデータ・レジスタを用いて指定することができます．

ASR #<即値>，Dy……Dyの内容を#<即値>のビット数だけ右へ算術シフトする．

ASR Dx,Dy……Dyの内容をDxの内容のビット数だけ右へ算術シフトする．

ASR <EA>……メモリ<EA>の内容を1ビットだけ右へ算術シフトする．

コンディション・コードの変化は，CとXに最下位ビット（LSB）がセットされ，Vフラグはシフト中に符号ビットの変化はありませんから常にゼロとなります．

NとZフラグは，おのおの結果が負，ゼロになれば1にセットされ，それ以外のときはゼロ・クリアされます．マシン語フォーマットは次のとおりです．

■レジスタ内容をシフトする場合：

15	14	13	12	11～9	8	7～6	5	4	3	2～0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	0	0	レジスタ

■メモリ内容をシフトする場合：

15	14	13	12	11	10	9	8	7	6	5～0
1	1	1	0	0	0	0	0	1	1	実効アドレス

次に，以上の命令のマシン語プログラミングの例題を考えてみましょう．

3.17　シフト命令のマシン語プログラミング例

例題15　$X=\frac{A+8*B}{2}$のプログラムを作り，これをマシン語に変換しなさい．

解き方

シフト命令を使う例題です．1ビット左へシフトすれば2倍になり，逆に1ビット右へシフトすれば1/2になることを利用すると，**図3.12**のようなプログラムができます．

```
        ORG       $3000
        MOVE.W    B, D0
        LSL.W     #3, D0
        MOVE.W    A, D1
        ADD.W     D1, D0
        MOVE.W    #1, D1
        LSR.W     D1, D0
        MOVE.W    D0, X
        TRAP      #13
A       DC.W      10
B       DC.W       5
X       DC.W       0
        END
```

図3.12　$X=\frac{A+8*B}{2}$のプログラム

それでは，これをマシン語に変換してみましょう．**MOVE.W B, D0**のマシン語は，**MOVE**命令のマシン語フォーマット；

15	14	13 12	11 9	8 6	5 3	2 0
0	0	サイズ	デスティネーション (レジスタ)	(モード)	ソース (モード)	(レジスタ)

に，サイズ＝ワード・オペレーション＝▼11▼，デスティネーションはD0で，アドレッシング・モードはデータ・レジスタ直接，したがってレジスタ＝▼000▼，モード・フィールド＝▼000▼，ソースはBでアブソリュート・ロングのアドレッシング・モードにすると，モード・フィールド＝▼111▼，レジスタ・フィールド＝▼001▼となり，以上を代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	1	………3039H

となり，これがマシン語の第1ワードとなります．これに続いてソースBの実効アドレスがきて，Bのアドレスはロング・ワードで▼00003020▼ですから，これをまとめると，

303900003020H

となり，これが**MOVE.W　B, D0**のマシン語となります．

LSL.W　#3, D0のマシン語フォーマットは，

15	14	13	12	11 9	8	7 6	5	4	3	2 0
1	1	1	0	カウント/レジスタ	1	サイズ	i/r	0	1	レジスタ

で，シフトするビット数に即値データを用いますから，i/r＝0，カウント/レジスタ・フィールド＝カウント数の即値＝#3＝▼011▼，サイズ・フィールド＝ワード・オペレーション＝▼01▼，レジスタ・フィールド＝データ・レジスタのNo.＝▼000▼，以上を代入して，

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	1	0	0	1	1	1	0	1	0	0	1	0	0	0	………E748H

E748Hが**LSL.W　#3,D0**のマシン語となります。

MOVE.W　A,D1のマシン語は最初の命令と同様にして，**32390000301EH**となります。

ADD.W　D1,D0のマシン語フォーマットは，

15	14	13	12	11　　9	8　　6	5　　　　0
1	1	0	1	レジスタ	OPモード	実効アドレス

で，レジスタ・フィールド＝D０＝▼000▼，オペレーション・モード＝ワードで，デスティネーションはD０＝▼001▼，実効アドレス・フィールド＝D１レジスタ直接＝▼000001▼を代入して，

15	14	13	12	11　　9	8　　6	5　　　　0	
1	1	0	1	0　0　0	0　0　1	0　0　0　0　0　1	………D041H

D041Hが**ADD.W　D1,D0**のマシン語となります。

MOVE.W　#1,D1のマシン語は，**MOVE**のマシン語フォーマットに，サイズ＝ワード＝▼11▼，デスティネーションのレジスタ＝D１＝▼001▼，モード＝データ・レジスタ直接＝▼000▼，ソース＝即値＝▼111100▼を代入して，

15	14	13	12	11　　9	8　　6	5　　3	2　　0	
0	0	1	1	0　0　1	0　0　0	1　1　1	1　0　0	………323CH

323CHが第１ワードとなり，これに即値（ワード）を含めると，

323C0001H

がマシン語となります。

LSR.W　D1,D0のマシン語は，**LSR**のマシン語フォーマット；

15	14	13	12	11　　9	8	7　6	5	4	3	2　　0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	0	1	レジスタ

に，カウント/レジスタ・フィールド＝D１＝▼001▼，サイズ＝ワード＝▼01▼，i/rフィールド＝レジスタ指定＝▼１▼，レジスタ＝D０＝▼000▼を代入して，

15	14	13	12	11			9	8	7	6	5	4	3	2		0	
1	1	1	0	0	0	1	0	0	1	1	0	1	0	0	0		………E268H

E268Hがマシン語となります．

MOVE.W　D0,Xは同じ**MOVE**命令のマシン語フォーマットに代入して，**33C000003022H**が得られますから，以上をまとめると**表3.11**のようなリストが得られます．

表3.11　例題15のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	303900003020		MOVE.W	B, D0
003006	E748		LSL.W	#3, D0
003008	32390000301E		MOVE.W	A, D1
00300E	D041		ADD.W	D1, D0
003010	323C0001		MOVE.W	#1, D1
003014	E268		LSR.W	D1, D0
003016	33C000003022		MOVE.W	D0, X
00301C	4E4D		TRAP	#13
00301E	000A	A	DC.W	10
003020	0005	B	DC.W	5
003022	0000	X	DC.W	0
			END	

3.18　回転（ローテート）命令

ローテート命令には，ROL，ROR，ROXL，ROXRの4つの命令があります．ROL，ROR命令は，おのおの左，右にオペランドの内容をローテートする命令で，ROXL，ROXR命令はX（拡張）フラグも含めて，おのおの左，右にローテートする命令です．次に，これらローテート命令の動作，機能，マシン語を詳しくみてみましょう．

3.18.1　ROL命令のマシン語

ROL(ROtate Left)命令は，オペランド内容を図3.13のように**左へ回転**します．そして，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容を左へ回転（ローテート）し，最上位ビットはC（キャリーフラグ）と最下位ビット(LSB)に入ります．X（拡張フラグ）はこの命令には関係なく，変化しません．

データ・レジスタ内容を回転する場合に，回転（ローテート）するビット数は即値（1～8の範囲）で指定することも，またデータ・レジスタを用いて指定することもできます(データ・レジスタを用いる場合，下位6ビット(0～63)

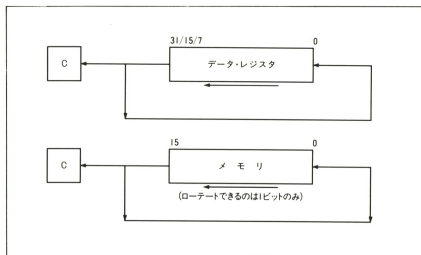



図3.13　ROL命令

が有効です).

前記の図3.13に示すように，データ・レジスタの左への回転は，32ビット，16ビット，8ビットの単位で行なうことができます．すなわち，ロング・ワード，ワード，バイトのオペレーションが可能ですが，メモリの場合はワード・オペレーションのみで，他を指定することはできません．さらに，メモリで回転することのできるビット数は，1ビットだけで，複数ビット回転する命令は用意されていません．

ROL　#<即値>，Dy ………Dyの内容を#<即値>のビット数だけ左へ回転する．

ROL　Dx，Dy………………Dyの内容をDxの内容のビット数だけ左へ回転する．

ROL　<EA>…………………メモリ<EA>の内容を1ビットだけ左へ回転する．

X	N	Z	V	C
—	*	*	0	*

コンディション・コードは，Cに最上位ビット(MSB)がセットされますが，回転するビット数が0のときはCはゼロ・クリアされます．また，Xは変化しないで，前のままの状態を保持します．Vフラグは常に0で，NとZはおのおの負，ゼロでセットされ，それ以外のときはゼロ・クリアされます．

マシン語フォーマットは，データ・レジスタ内容を回転する場合と，メモリ内容を回転する場合と2とおり用意されています．

■レジスタ内容を回転する場合：

15	14	13	12	11　　9	8	7　6	5	4	3	2　　0
1	1	1	0	カウント／レジスタ	1	サイズ	i/r	1	1	レジスタ

カウント／レジスタ・フィールドは，i/r＝0のときは回転（ローテート）するカウント数（ビット数）が即値で，このフィールドにセットされます．0は8を，また1～7は1～7を表わします．i/r＝1のときは，回転するビット数はデータ・レジスタ内にあり，そのデータ・レジスタのNo.がこのカウント／レ

ジスタ・フィールドにセットされます.

サイズ・フィールドは，オペレーションのサイズを指定するところで，

００……バイト・オペレーション

０１……ワード・オペレーション

１０……ロング・ワード・オペレーション

となります.

i/rフィールドは，回転(ローテート)するビット数に即値データを用いるか，あるいはデータ・レジスタを用いるかを指定する１ビットのフィールドで，

i/r＝０……即値データ

i/r＝１……データ・レジスタ

が指定されます.

レジスタ・フィールドは，回転（ローテート）されるデータが格納されているデータ・レジスタのNo.を指定します.

■メモリ内容を回転する場合：

15	14	13	12	11	10	9	8	7	6	5 … 0
1	1	1	0	0	1	1	1	1	1	実効アドレス

実効アドレス・フィールドは，回転（ローテート）するメモリのアドレッシング・モードを指定します.

3.18.2　ROR命令のマシン語

ROR(ROtate Right)命令は，オペランド内容を図3.14のように**右へ回転**します．そして，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容を右へ回転（ローテート）し，最下位ビット(LSB)はC（キャリーフラグ）と最上位ビット(MSB)に入ります．X（拡張フラグ）は，この命令に関係なく変化しません.

データ・レジスタ内容を回転する場合，回転するビット数の指定は，**ROL**命令と同様に，即値（１～８)でもデータ・レジスタを用いても行なうことができます．また，バイト，ワード，ロング・ワードのオペレーションが可能なのも同様です.

これに対して，メモリ内容を回転する場合，**ROL**命令と同じように１ビット

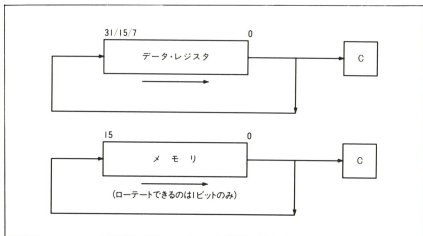



図3.14　ROR命令

の回転しかできなく，それもワード・オペレーションが許されるのみです．

ROR　#〈即値〉，Dy　………Dyの内容を#〈即値〉のビット数だけ右へ回転する．

ROR　Dx，Dy　……………Dyの内容をDxの内容のビット数だけ右へ回転する．

ROR　〈EA〉…………………メモリ〈EA〉の内容を1ビットだけ右へ回転する．

コンディション・コードの変化は，Cに最下位ビット(LSB)がセットされる点を除いて，**ROL**命令の場合と同様です．

マシン語フォーマットは，データ・レジスタ内容を回転する場合と，メモリ内容を回転する場合と2とおり用意されており，おのおの次のとおりです．

■レジスタ内容を回転する場合：

15	14	13	12	11　9	8	7　6	5	4	3	2　0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	1	1	レジスタ

カウント／レジスタ・フィールド，サイズ・フィールド，i/rフィールド，レ

ジスタ・フィールドの指定の仕方は，ROL命令のときとまったく同様です．

■メモリ内容を回転する場合：

15	14	13	12	11	10	9	8	7	6	5　　　　0
1	1	1	0	0	1	1	0	1	1	実効アドレス

実効アドレス・フィールドも，ROL命令の場合と同じ指定の仕方です．

3.18.3　ROXL命令のマシン語

ROXL(ROtate through X Left)命令は，オペランド内容を，図3.15のように**X（拡張フラグ）を含めて左へ回転**します．そしてデスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容を，X(拡張フラグ)を含めて左へ回転（ローテート）し，最上位ビット(MSB)はC（キャリーフラグ）とX（拡張フラグ）に入り，Xは最下位ビット(LSB)に入ります．

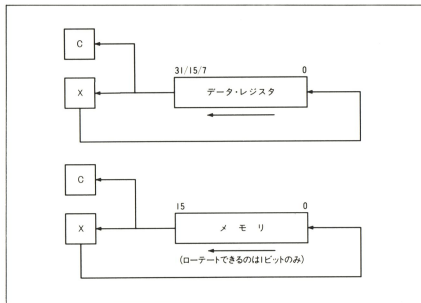



図3.15　ROXL命令

データ・レジスタ内容を，Xを含めて回転する場合は，回転（ローテート）するビット数を即値（1～8の範囲）で指定することも，またデータ・レジスタを用いて指定することもできます（データ・レジスタを用いる場合，下位6ビット（0～63）が有効）．

データ・レジスタのX（拡張フラグ）を含めて左への回転は，32ビット，16ビット，8ビットの単位で行なうことができます．すなわち，バイト，ワード，ロング・ワードのオペレーションが可能です．

メモリ内容をXを含めて回転する場合は，ワード・オペレーションのみが指定でき，回転することのできるビット数は1ビットだけです．

ROXL　#<即値>，Dy………Dyの内容をXフラグを含めて#<即値>のビット数だけ左へ回転する．

ROXL　Dx，Dy……………Dyの内容をXフラグを含めてDxの内容のビット数だけ左へ回転する．

ROXL　<EA>………………メモリ<EA>の内容をXフラグを含めて1ビットだけ左へ回転する．

X	N	Z	V	C
*	*	*	0	*

コンディション・コードは，Xに最上位ビット（MSB）がセットされますが，回転するビット数が0のときは，変化しないで前のままの状態を保持します．また，Cにも最上位ビット（MSB）がセットされ，回転するビット数が0のときは，Xと同じ値にセットされます．Vフラグは常に0で，NとZはおのおの負，ゼロでセットされ，それ以外のときはゼロ・クリアされます．

マシン語フォーマットは次のとおりです．

■**レジスタ内容をXを含めて回転する場合**：

15	14	13	12	11　　9	8	7　6	5	4	3	2　　0
1	1	1	0	カウント/レジスタ	1	サイズ	i/r	1	0	レジスタ

（各フィールドの指定の仕方は**ROL**命令とまったく同様です）

■メモリ内容をXを含めて回転する場合：

15	14	13	12	11	10	9	8	7	6	5　　　　0
1	1	1	0	0	1	0	1	1	1	実効アドレス

（実効アドレス・フィールドはメモリのアドレッシング・モードを指定）

3.18.4　ROXR命令のマシン語

ROXR(ROtate through X Right)命令は，オペランド内容を，**図3.16**のように**X(拡張フラグ)を含めて右へ回転**します．そして，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容を，X（拡張フラグ）を含めて右へ回転（ローテート）し，最下位ビット(LSB)はC（キャリーフラグ）とX（拡張フラグ）に入り，Xは最上位ビット(MSB)に入ります．

データ・レジスタ内容を，Xを含めて回転する場合，回転するビット数は前述

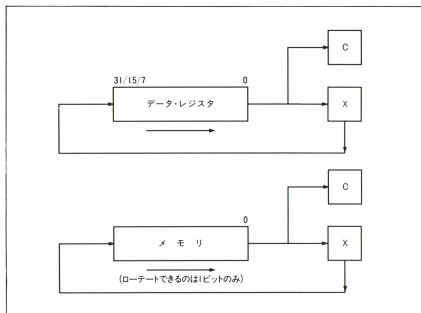



図3.16　ROXR命令

したローテート命令と同様に，即値を用いても，データ・レジスタを用いても指定することができます．

データ・レジスタ内容の回転はバイト，ワード，ロング・ワード・オペレーションが可能ですが，メモリ内容の回転では，ワード・オペレーションのみが許され，回転のビット数も，1ビットのみとなるのも前の命令と同じです．

ROXR　#<即値>，Dy ……Dyの内容をXフラグを含めて#<即値>のビット数だけ右へ回転する．

ROXR　Dx，Dy……………Dyの内容をXフラグを含めてDxの内容のビット数だけ右へ回転する．

ROXR　<EA>………………メモリ<EA>の内容をXフラグを含めて1ビットだけ右へ回転する．

コンディション・コードの変化は，XとCに最下位ビット(LSB)がセットされ，回転するビット数が0のときは，Xは変化せず，CもXと同じ値となります．他は**ROXL**命令の場合と同様です．

マシン語フォーマットは次のとおりです．

■レジスタ内容をXを含めて回転する場合：

15	14	13	12	11　9	8	7　6	5	4	3	2　0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	1	0	レジスタ

(各フィールドの指定の仕方は**ROL**命令と同様です)

■メモリ内容をXを含めて回転する場合：

15	14	13	12	11	10	9	8	7	6	5　0
1	1	1	0	0	1	0	0	1	1	実効アドレス

(実効アドレス・フィールドはメモリのアドレッシング・モードを指定)

次に，以上の命令のマシン語プログラミング，マシン語変換の例題を行なうことにしましょう．

3.19 回転(ローテート)命令のマシン語プログラミング例

例題16 次のプログラムのマシン語を作りなさい.

```
        ORG      $3000
        ROL      D1, D0
        ROR.B    D3, D2
        ROR.L    D5, D4
        ROL      #5, D0
        ROL      -(A0)
        ROR      (A1)
        ROL      $3020
        ROR      A10
A10     DC.W     3
        END
```

解き方

ROL D1, D0のマシン語は, **ROL**命令のマシン語フォーマット;

15	14	13	12	11 〜 9	8	7 〜 6	5	4	3	2 〜 0
1	1	1	0	カウント/レジスタ	1	サイズ	i/r	1	1	レジスタ

に, カウント/レジスタ・フィールドはデータ・レジスタD1を用いて回転ビット数を指定しますから, データ・レジスタNo.の▼001▼を, サイズ・フィールドはワード・オペレーションで▼01▼, またi/rフィールドはデータ・レジスタを用いて, 回転ビット数を指定しますからi/r=1, レジスタ・フィールドは回転するデータ・レジスタのNo.=D0=▼000▼となり, 以上から,

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	1	0	0	0	1	1	0	1	1	1	1	0	0	0	………E378H

となり，**E378H**が**ROL　D1，D0**のマシン語となります．

ROR.B　D3，D2のマシン語は，**ROR**命令のマシン語フォーマット；

15	14	13	12	11		9	8	7	6	5	4	3	2		0
1	1	1	0	カウント/レジスタ			0	サイズ		i/r	1	1	レジスタ		

に，カウント／レジスタ・フィールド＝D3のレジスタNo.＝▼011▼，サイズ＝バイト・オペレーション＝▼00▼，i/rフィールドはデータ・レジスタを用いていますからi/r＝１，レジスタ・フィールド＝D2レジスタのNo.＝▼010▼をセットして，

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	………E63AH

となり，**E63AH**が**ROR.B　D3，D2**のマシン語となります．

ROR.L　D5, D4のマシン語は，同じマシン語フォーマットにカウント／レジスタ＝D5＝▼101▼，サイズ＝ロング・オペレーション＝▼10▼，i/r＝レジスタ＝▼1▼，レジスタ・フィールド＝D4レジスタ＝▼100▼を代入して，

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	1	0	1	0	1	0	1	0	1	1	1	1	0	0	………EABCH

EABCHが**ROR.L　D5，D4**のマシン語となります．

ROL　#5, D0のマシン語は，**ROL**命令のマシン語フォーマットに，カウント／レジスタ・フィールド＝カウントの即値＝#５＝▼101▼，サイズ・フィールド＝ワード・オペレーション＝▼01▼，i/rフィールドは即値を用いて回転ビット数を指定しますから，i/r＝▼０▼，レジスタ・フィールド＝回転するデータ・レジスタNo.（D0）＝▼000▼をセットして，

15	14	13	12	11	9	8	7	6	5	4	3	2　　0	
1	1	1	0	1	0	1	1	0	1	0	1	1	0 0 0	………EB58H

EB58HがROL　#5, D0のマシン語となります．

ROL　−(A0)のマシン語は，メモリ内容をROLする命令のマシン語フォーマット；

15	14	13	12	11	10	9	8	7	6	5　　　　0
1	1	1	0	0	1	1	1	1	1	実効アドレス

を用いてマシン語を作ります．

実効アドレス・フィールドは，A0レジスタを用いる“プリ・デクリメント・アドレス・レジスタ間接”のアドレッシング・モードで，表3.3より▼100000▼となります．

A0レジスタを表わす

したがって，マシン語は，

表3.3　モード，レジスタ・フィールドによって決められるアドレッシング・モード

モード・フィールド	レジスタ・フィールド	アドレッシング・モード
0 0 0	Dn	データ・レジスタ直接
0 0 1	An	アドレス・レジスタ直接
0 1 0	An	アドレス・レジスタ間接
0 1 1	An	ポスト・インクリメント・アドレス・レジスタ間接
1 0 0	An	プリ・デクリメント・アドレス・レジスタ間接
1 0 1	An	ディスプレースメント付アドレス・レジスタ間接
1 1 0	An	インデックス，ディスプレースメント付アドレス・レジスタ間接
1 1 1	0 0 0	アブソリュート・ショート
1 1 1	0 0 1	アブソリュート・ロング
1 1 1	0 1 0	ディスプレースメント付プログラム・カウンタ相対
1 1 1	0 1 1	インデックス，ディスプレースメント付プログラム・カウンタ相対
1 1 1	1 0 0	即　値

E7E0Hが**ROL　－(A0)**のマシン語となります．

前述したように，メモリの回転はワード・オペレーションだけで，しかも回転できるビット数は，１ビットのみです(注：68000では，メモリ内容の回転は１ビットしかできませんが，8086では，CLレジスタを用いて複数ビットの回転が可能で，しかもワード・オペレーションとバイト・オペレーションができました．メモリ回転では，68000の能力は低いといえます)．

ROR　(A1)のマシン語は，メモリを**ROR**する命令のマシン語フォーマット；

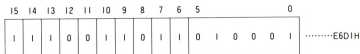
15	14	13	12	11	10	9	8	7	6	5　　　　0
1	1	1	0	0	1	1	0	1	1	実効アドレス

を使ってマシン語を生成します．

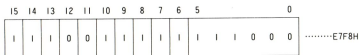
実効アドレス・フィールドは，A1レジスタを用いる“アドレス・レジスタ間接”アドレッシング・モードで，表３．３より▼010001▼となります．

（001：A1レジスタを表わす）

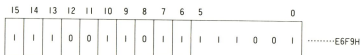
マシン語は，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	1	0	0	1	1	0	1	1	0	1	0	0	0	1	………E6D1H

E6D1Hが**ROR　(A1)**のマシン語となります．

ROL　$3020のマシン語は，メモリ内容を**ROL**する命令のマシン語フォーマットに，実効アドレス・フィールド＝アブソリュート(絶対)ショートのアドレッシング・モード＝▼111000▼を代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	………E7F8H

E7F8Hとなります．ただし，これはマシン語の最初の１ワードで，次にショートで絶対アドレス3020Hが続きます．以上から，**ROL　$3020**のマシン語は**E7F83020H**となります．

ROR　A10のマシン語は，メモリ内容を**ROR**する命令のマシン語フォーマットに，実効アドレス・フィールド＝アブソリュート(絶対)ロングのアドレッシング・モード＝▼111001▼をセットして，

15	14	13	12	11	10	9	8	7	6	5					0	
1	1	1	0	0	1	1	0	1	1	1	1	1	0	0	1	………E6F9H

E6F9Hとなり，これが最初の1ワードで次に絶対ロングのアドレス▼00003016▼が続きます．

したがって，**ROR　A10**のマシン語は**E6F900003016H**となります．

これらをまとめると，**表3.12**のようなハンド・アセンブルのリストを得ることができます．

次に，**例題17**を見てください．

表3.12　例題16のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	E378		ROL	D1, D0
003002	E63A		ROR.B	D3, D2
003004	EABC		ROR.L	D5, D4
003006	EB58		ROL	#5, D0
003008	E7E0		ROL	−(A0)
00300A	E6D1		ROR	(A1)
00300C	E7F83020		ROL	$3020
003010	E6F900003016		ROR	A10
003016	0003	A10	DC.W	3
			END	

例題17　次のプログラムのマシン語を作りなさい．

```
        ORG     $3000
        ROXL    D1, D0
        ROXR.B  D3, D2
        ROXR.L  D5, D4
        ROXL    #4, D0
        ROXL    -(A0)
        ROXR    (A1)
        ROXL    $3020
        ROXR    A20
A20     DC.W    5
        END
```

解き方

ROXL命令は，レジスタ内容をXを含めて回転する場合と，メモリ内容をXを含めて回転する場合との２とおりのマシン語フォーマットがあります．**ROXL D1，D0**命令は，レジスタ内容の回転ですから，次のマシン語フォーマットが用いられます．

レジスタ内容をXを含めて回転する場合：

15	14	13	12	11　9	8	7　6	5	4	3	2　0
1	1	1	0	カウント/レジスタ	1	サイズ	i/r	1	0	レジスタ

回転するカウント・ビット数は，データ・レジスタに保持されますから，i/rフィールドはi/r＝１となります．また，カウント／レジスタ・フィールドはi/r＝１のとき，回転するカウント・ビット数を保持するデータ・レジスタの番号がセットされ，D1のレジスタ番号＝１をセットします．

レジスタ・フィールドは，回転するデータを保持するデータ・レジスタの番号がセットされ，D0のレジスタ番号＝０をセットします．サイズは，オペレーションのサイズを指定するフィールドで，ワード・オペレーション＝▼01▼を

セットします．これらをROXL命令のマシン語フォーマットに代入して，

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	1	0	0	0	1	1	0	1	1	1	0	0	0	0	………E370H

E370Hが　**ROXL　D1，D0**のマシン語となります．

ROXR.B　D3，D2のマシン語は，**ROXR**命令のレジスタ内容を回転するケースのマシン語フォーマット；

レジスタ内容をXを含めて回転する場合：

15	14	13	12	11		9	8	7	6	5	4	3	2		0
1	1	1	0	カウント/レジスタ			0	サイズ		i/r	1	0	レジスタ		

に，カウント／レジスタ・フィールド＝回転するカウント・ビット数を保持するデータ・レジスタ番号＝３＝▼011▼，サイズ・フィールド＝バイト・オペレーション＝▼00▼，i/r＝回転カウント・ビット数をデータ・レジスタに保持＝１，レジスタ・フィールド＝回転データが入っているデータ・レジスタのレジスタ番号＝２＝▼010▼を代入して，

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0	………E632H

E632Hが　**ROXR.B　D3，D2**のマシン語となります．

ROXR.L　D5，D4命令のマシン語は，オペレーション・サイズがロング・ワードで，データ・レジスタにD5，D4が使用される点を除いて，前命令と同じに考えればよく，

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	1	0	1	0	1	0	1	0	1	1	0	1	0	0	………EAB4H

EAB4Hがそのマシン語となります．

ROXL　#4，D0　命令では，回転するカウント・ビット数が即値で与えられているケースで，**ROXL**のマシン語フォーマット；

15	14	13	12	11　　9	8	7　6	5	4	3	2　　0
1	1	1	0	カウント/レジスタ	1	サイズ	i/r	1	0	レジスタ

において，i/rフィールドは即値を用いるため，i/r＝0となります．i/r＝0のとき，カウント／レジスタ・フィールドは即値の＃4がセットされます．サイズはワード・オペレーションで▼01▼，レジスタ・フィールドはD0で▼000▼となりますから，

15	14	13	12	11　　9	8	7　6	5	4	3	2　　0	
1	1	1	0	1　0　0	1	0　1	0	1	0	0　0　0	………E950H

E950Hが即値を用いた**ROXL　#4，D0**命令のマシン語となります．

ROXL　—(A0)命令は，メモリ内容の回転ですから**ROXL**のマシン語フォーマットのうち，次のように「メモリ内容をXを含めて回転する場合」のフォーマットを用います．

メモリ内容をXを含めて回転する場合：

15	14	13	12	11	10	9	8	7	6	5　　　　0
1	1	1	0	0	1	0	1	1	1	実効アドレス

ここで実効アドレス・フィールドは，メモリ〈EA〉のアドレッシング・モードで，—(A0)は▼100000▼となりますから，これを代入して，

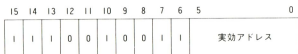
15	14	13	12	11	10	9	8	7	6	5　　　　0	
1	1	1	0	0	1	0	1	1	1	1　0　0　0　0　0	………E5E0H

E5E0Hが　**ROXL　—(A0)**のマシン語となります．

メモリ内容の回転のマシン語フォーマットには，サイズ・フィールドやカウント・ビット数のフィールドがありませんが，これは，メモリ内容の回転は1ビットのみ可能で，しかもオペレーションは常にワード・オペレーションであるためです．

ROXR　(A1)の命令のマシン語フォーマットは，

メモリ内容をXを含めて右へ回転する場合：

15	14	13	12	11	10	9	8	7	6	5　　　　　　0
1	1	1	0	0	1	0	0	1	1	実効アドレス

で，実効アドレス・フィールドに(A1)＝▼010001▼を代入して，

15	14	13	12	11	10	9	8	7	6	5					0	
1	1	1	0	0	1	0	0	1	1	0	1	0	0	0	1	………E4D1H

E4D1Hがマシン語となります．

ROXL　$3020命令のマシン語は，メモリ回転のマシン語フォーマット；

メモリ内容をXを含めて左へ回転する場合：

15	14	13	12	11	10	9	8	7	6	5　　　　　　0
1	1	1	0	0	1	0	1	1	1	実効アドレス

に，実効アドレス＝アブソリュート・ショート＝▼111000▼を代入し，

15	14	13	12	11	10	9	8	7	6	5					0	
1	1	1	0	0	1	0	1	1	1	1	1	1	0	0	0	………E5F8H

となり，**E5F8H**が第1ワードとなり，次の第2ワードに**$3020**が続きます．したがって，**ROXL　$3020**のマシン語は，

E5F83020H

となります．

ROXR　A20のマシン語は，**ROXR**のメモリ回転のマシン語フォーマット；

メモリ内容をXを含めて右へ回転する場合：

15	14	13	12	11	10	9	8	7	6	5 ... 0
1	1	1	0	0	1	0	0	1	1	実効アドレス

に，実効アドレス＝アブソリュート・ロング＝▼111001▼を代入して，

15	14	13	12	11	10	9	8	7	6	5					0	
1	1	1	0	0	1	0	0	1	1	1	1	1	0	0	1	………E4F9H

E4F9Hが，**ROXR　A20**のマシン語の第１ワードとなります．次にA20のアドレス▼00003016▼が続くから，**ROXR　A20**のマシン語は，

E4F900003016H

となります．

以上をまとめると，**表３.13**のようなハンド・アセンブル・リストになります．

表３.13　例題17のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	E370		ROXL	D1, D0
003002	E632		ROXR.B	D3, D2
003004	EAB4		ROXR.L	D5, D4
003006	E950		ROXL	#4, D0
003008	E5E0		ROXL	−(A0)
00300A	E4D1		ROXR	(A1)
00300C	E5F83020		ROXL	$3020
003010	E4F900003016		ROXR	A20
003016	0005	A20	DC.W	5
			END	

3.20　ビット操作命令

16ビット・マイコン68000の命令のなかで強力なものの1つに，ビット操作命令があげられます．他の16ビット・マイコンには，この命令を含まないものが多いようです(たとえば，8086にはこのビット操作命令がありません．さらに，80186,80286になってもビット操作命令はサポートされておらず，ビット操作に関しては8086,80286は貧弱であるといえます)

ビット操作命令は，読んだとおりの意味で，ビットを操作する命令で，レジスタやメモリの特定のビットを調べたり，そのビットに値をセットしたりする命令です．ビット操作命令には，**BTST**，**BSET**，**BCLR**，**BCHG**の4つの命令があります．

BTST命令はビットをテストする命令，**BSET**命令はビットをセットする命令，**BCLR**命令はビットをクリアする命令，**BCHG**命令はビットを反転する命令です．次に，これらビット操作命令の動作，機能，マシン語を詳しく見ることにしましょう．

3.20.1　BTST命令のマシン語

BTST（**Bit TeST**）命令は，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容のソース・オペランドで指定されるビットをテストして，0か1かによっておのおのZフラグがセットされたり，リセットされたりします．すなわち，指定されたビットの値が0であれば，Zフラグは1になり，ビットの値が1であれば，Zフラグは0になります．

ビットをテストして，それが1であるか0であるかを調べるだけの命令が，ビット・テスト**BTST**命令です．デスティネーション・オペランドにデータ・レジスタを用いた場合，ロング・ワード・オペレーションとなり，ビット番号には0から31（0～31）が用いられます．メモリを用いた場合は，バイト・オペレーションとなり，ビット番号には0から7（0～7）まで用いられます．

ビット番号の指定の仕方に2とおりあり，1つは即値で直接指定する方法，他はデータ・レジスタを用いて，そこにビット番号を設定して指定する方法です．デスティネーション・オペランドがデータ・レジスタの場合は，0から31までのビット番号を表わすのに下位5ビットを用い，メモリの場合は，0から7までのビット番号を表わすのに下位3ビットを用います．

BTST　Dn, 〈EA〉 ………………〈EA〉の内容のDnで指定されるビットをテストする．結果がZフラグに影響する．

BTST　#〈即値〉, 〈EA〉 …………〈EA〉の内容の #〈即値〉で指定されるビットをテストする．結果がZフラグに影響する．

X	N	Z	V	C
—	—	*	—	—

コンディション・コードは，前述したようにZフラグにテスト結果が反映し他のフラグは変化しません．テストされたビットの値が0のとき，Zフラグは1にセットされ，値が1のときはZフラグは0にリセットされます．

マシン語フォーマットは，ビット番号をデータ・レジスタを用いて指定する場合と，即値で直接指定する場合と2とおり用意されています．

■ビット番号をデータ・レジスタを用いて指定する場合：

15	14	13	12	11　　9	8	7	6	5　　0
0	0	0	0	レジスタ	1	0	0	実効アドレス

レジスタ・フィールドは，ビット番号を入れるデータ・レジスタのNo.がセットされ，実効アドレス・フィールドはデスティネーション・オペランドのアドレッシング・モードを指定します．

■ビット番号を即値で直接指定する場合：

実効アドレス・フィールドは，デスティネーション・オペランドのアドレッシング・モードを指定し，ビット番号フィールドはビット番号が即値で，このフィールドにセットされます．

3.20.2　BSET命令のマシン語

BSET（**Bit test and SET**）命令は，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容のソース・オペランドで指定されるビットをテストし，その後そのビットを1にセットする命令です．すなわち，指定されたビットをテストし，そのビットの値が0であれば，Zフラグを1にセットし，1であればZフラグを0にゼロ・クリアして，Zフラグにテスト結果を反映してから，そのテストしたビットを▼1▼にセットします．

BTST命令の場合と同様に，デスティネーション・オペランドにデータ・レジスタを用いたときは，32ビットのロング・ワードの動作となり，ビット番号には0～31が使用され，メモリを用いたときは8ビットのバイトの動作となり，ビット番号には0～7が使用されます．ビット番号の指定の仕方も，**BTST**命令の場合と同様に2とおりあり，1つは即値で直接指定する方法，他はデータ・レジスタを用いる方法です．

BSET　Dn，<EA> ……………………<EA>の内容のDnで指定されるビットをテストし，その結果をZフラグに反映させ，そのビットを1にセットする．

BSET　#<即値>，<EA> …………<EA>の内容の#<即値>で指定されるビットをテストし，その結果をZフラグに反映させ，そのビットを1にセットする．

X	N	Z	V	C
—	—	*	—	—

コンディション・コードは，**BTST**命令の場合と同様に，Zフラグにテスト結果が反映し，他のフラグは変化しません．ビットの値が0のとき，Zフラグは1にセットされ，値が1のときはZフラグは0にリセットされます．

マシン語フォーマットは，**BTST**命令と同様に，ビット番号をデータ・レジスタを用いて指定する場合と，即値で直接指定する場合と2とおり用意されてい

ます．

■ビット番号をデータ・レジスタを用いて指定する場合：

15	14	13	12	11　　9	8	7	6	5　　0
0	0	0	0	レジスタ	1	1	1	実効アドレス

レジスタ・フィールドは，ビット番号が格納されるデータ・レジスタNo.を指定し，実効アドレス・フィールドはデスティネーション・オペランドのアドレッシング・モードを指定します．

■ビット番号を即値で直接指定する場合：

実効アドレス・フィールドは，デスティネーション・オペランドのアドレッシング・モードを指定し，ビット番号フィールドはビット番号が即値で，このフィールドにセットされます．

3.20.3　BCLR命令のマシン語

BCLR（Bit test and CLeaR）命令は，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容のソース・オペランドで指定されるビットをテストし，その後，そのビットを0にクリアする命令です．指定されたビットをテストし，そのビットの値が0であれば，Zフラグを1にセットし，値が1であればZフラグを0にゼロ・クリアして，Zフラグにテスト結果を反映してから，そのテストしたビットを0にゼロ・クリアします．

BCLR　Dn,　〈EA〉 ………………〈EA〉の内容のDnで指定されるビットをテストし，その結果をZフラグに反映

させ，そのビットを0にクリアする．

BCLR　#〈即値〉，〈EA〉　…………〈EA〉の内容の#〈即値〉で指定されるビットをテストし，その結果をZフラグに反映させ，そのビットを0にクリアする．

X	N	Z	V	C
—	—	*	—	—

コンディション・コードの変化は，BTST，BSET命令とまったく同様で，Zフラグのみ影響を受けます．

マシン語フォーマットを次に示します．

■ビット番号をデータ・レジスタを用いて指定する場合：

レジスタ・フィールド，実効アドレス・フィールドはBTST，BSET命令と同じに用いられます．

■ビット番号を即値で直接指定する場合：

実効アドレス・フィールド，ビット番号フィールドはBTST，BSET命令の場合とまったく同様に用いられます．

3.20.4　BCHG命令のマシン語

BCHG（Bit test and CHanGe）命令は，デスティネーション・オペランドで指定したデータ・レジスタ，またはメモリの内容のソース・オペランドで指定されるビットをテストし，その後そのビットの値を反転する命令です．指定されたビットをテストし，そのビットの値が０であれば，Zフラグを１にセットし，値が１であればZフラグを０にゼロ・クリアして，Zフラグにテスト結果を反映してから，そのテストしたビットの値を反転します．すなわち，０であれば１に，１であれば０に反転します．

BCHG　Dn，〈EA〉…………〈EA〉の内容のDnで指定されるビットをテスをテストし，その結果をZフラグに反映そのビさせ，そのビットを１にセットする．

BCHG　#〈即値〉，〈EA〉……〈EA〉の内容の#〈即値〉で指定されるビットをテストし，その結果をZフラグに反映させ，そのビットの値を反転する．

コンディション・コードの変化は，**BTST**，**BSET**，**BCLR**命令とまったく同様で，Zフラグのみ影響を受けます．

マシン語フォーマットを次に示します．

■ビット番号をデータ・レジスタを用いて指定する場合：

15	14	13	12	11　　9	8	7	6	5　　　　0
0	0	0	0	レジスタ	1	0	1	実効アドレス

レジスタ・フィールド，実効アドレス・フィールドは**BTST**，**BSET**，**BCLR**命令と同様に用いられます．

■ビット番号を即値で直接指定する場合：

15	14	13	12	11	10	9	8	7	6	5 ～ 0
0	0	0	0	1	0	0	0	0	1	実効アドレス

15	14	13	12	11	10	9	8	7 ～ 0
0	0	0	0	0	0	0	0	ビット番号

実効アドレス・フィールド，ビット番号フィールドは**BTST**，**BSET**，**BCLR**命令とまったく同様に用いられます．

次に，ビット操作命令のマシン語プログラミング，マシン語変換の例題を行なってみましょう．

3.21　ビット操作命令のマシン語プログラミング例

例題18　次のプログラムのマシン語を作りなさい．

```
            ORG     $3000
            BTST    #7, STATUS
            BEQ.S   ABC1
            BRA     ABC2
ABC1        BSET    D0, D1
            BSET    #4, (A1)+
            BCLR    D0, D1
            BCLR    #7, (A2)
ABC2        BCHG    #7, STATUS
            BCHG    D0, D2
            TRAP    #13
STATUS      DC.B    0
            END
```

解き方

BTST #7, STATUSのマシン語は，2とおり用意されているBTST命令のマシン語フォーマットのうち，ビット番号を即値で直接指定するマシン語フォーマットを用います．

実効アドレス・フィールドはデスティネーション・オペランドのアドレッシング・モードを指定し，この命令ではアブソリュート(絶対)ロングのアドレッシング・モードですから，表3.3より▼111001▼となります．そしてこれをセットして，

表3.3 モード，レジスタ・フィールドによって決められるアドレッシング・モード

モード・フィールド	レジスタ・フィールド	アドレッシング・モード
0 0 0	Dn	データ・レジスタ直接
0 0 1	An	アドレス・レジスタ直接
0 1 0	An	アドレス・レジスタ間接
0 1 1	An	ポスト・インクリメント・アドレス・レジスタ間接
1 0 0	An	プリ・デクリメント・アドレス・レジスタ間接
1 0 1	An	ディスプレースメント付アドレス・レジスタ間接
1 1 0	An	インデックス，ディスプレースメント付アドレス・レジスタ間接
1 1 1	0 0 0	アブソリュート・ショート
1 1 1	0 0 1	アブソリュート・ロング
1 1 1	0 1 0	ディスプレースメント付プログラム・カウンタ相対
1 1 1	0 1 1	インデックス，ディスプレースメント付プログラム・カウンタ相対
1 1 1	1 0 0	即 値

15	14	13	12	11	10	9	8	7	6	5					0	
0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	1	………0839H

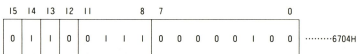
0839Hとなり，これが最初の１ワードで，次に即値でビット番号0007Hの１ワードが続き，最後に絶対ロングのアドレス▼00003026▼が続きます．以上から，

BTST　#7，STATUSのマシン語は，

0839000700003026H

となります．

BEQ.S　ABC1の条件ブランチのフォーマットは，

で，条件フィールドは**表3.14**よりEQ＝▼0111▼，DISP 8＝▼04H▼を代入して，

6704Hが**BEQ.S　ABC1**のマシン語となります．

BRA　ABC2の無条件ブランチのマシン語フォーマットは，

15	0
DISP16（このときDISP 8＝0）	

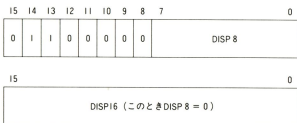



表3．14　ブランチ条件とコード

ニーモニック	ブランチ条件	コード
CC	キャリークリア(carry clear)	0100
CS	キャリーセット(carry set)	0101
EQ	等しい(equal)	0111
GE	大きい，または等しい(greater or equal)	1100
GT	大きい(greater)	1110
HI	高い(high)	0010
LE	小さい，または等しい(less or equal)	1111
LS	低い，または同じ(low or same)	0011
LT	小さい(less)	1101
MI	負(マイナス)(minus)	1011
NE	等しくない(not equal)	0110
PL	正(プラス)(plus)	1010
VC	オーバーフロー・クリア，オーバーフローなし (overflow clear, no overflow)	1000
VS	オーバーフロー・セット，オーバーフロー (overflow set, overflow)	1001

で，DISP16=▼000E▼となり，**BRA　ABC2**のマシン語は**6000000EH**となります．

ABC1　BSET　D0，D1のマシン語は，ビット番号を，データ・レジスタを用いて指定する**BSET**命令のマシン語フォーマット；

15	14	13	12	11　　9	8	7	6	5　　　0
0	0	0	0	レジスタ	1	1	1	実効アドレス

において，レジスタ・フィールド＝ビット番号が格納されているデータ・レジスタのNo.(D0)＝▼000▼，実効アドレス・フィールド＝デスティネーション・オペランドのアドレッシング・モード＝D1＝レジスタ直接＝▼000001▼を代入して，

15	14	13	12	11		9	8	7	6	5					0	
0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	………01C1H

01C1HがABC1　BSET　D0, D1のマシン語となります.

BSET　#4, (A1)＋のマシン語は, ビット番号を即値で直接指定するBSET命令のマシン語フォーマット；

において, 実効アドレス・フィールド＝デスティネーション・オペランドのアドレッシング・モード＝(A 1)＋＝▼011001▼, ビット番号フィールド＝▼0004H▼を代入して,

08D90004HがBSET　#4,(A1)＋のマシン語となります.

BCLR　D0, D1のマシン語は, ビット番号をデータ・レジスタを用いて指定するBCLR命令のマシン語フォーマット；

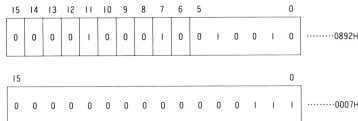
15	14	13	12	11　　9	8	7	6	5　　0
0	0	0	0	レジスタ	1	1	0	実効アドレス

において，レジスタ・フィールド＝D０＝▼000▼，実効アドレス・フィールド＝D１＝レジスタ直接＝▼000001▼を代入して，

15	14	13	12	11			9	8	7	6	5				0	
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	………0181H

0181HがBCLR　D0，D1のマシン語となります．

BCLR　#7，(A2) のマシン語は，ビット番号を即値で直接指定するBCLR命令のマシン語フォーマット；

において，実効アドレス・フィールド＝（A２）＝▼010010▼，ビット番号フィールド＝▼0007H▼を代入して，

08920007HがBCLR　#7，(A２）のマシン語となります．

ABC2　BCHG　#7，STATUSのマシン語は，２とおり用意されているBCHG命令マシン語フォーマットのうち，ビット番号を即値で直接指定するマシン語フォーマットを用います．

実効アドレス・フィールド＝アブソリュート(絶対)ロング＝▼111001▼となり，これをセットして，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	1	………0879H

0879Hとなり，これが最初の1ワードで，次に即値でビット番号0007Hのワードが続き，最後に絶対(アブソリュート)ロングのアドレス▼00003026▼がきます．
以上から，**ABC2　BCHG　#7，STATUS**のマシン語は，

0879000700003026H

となります．

BCHG　D0，D2のマシン語は，ビット番号をデータ・レジスタを用いて指定する**BCHG**命令のマシン語フォーマット；

15	14	13	12	11　9	8	7	6	5　0
0	0	0	0	レジスタ	1	0	1	実効アドレス

において，レジスタ・フィールド＝D0＝▼000▼，実効アドレス・フィールド＝D2＝レジスタ直接＝▼000010▼を代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	………0142H

表3.15 例題18のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	0839000700003026		BTST	#7, STATUS
003008	6704		BEQ.S	ABC1
00300A	6000000E		BRA	ABC2
00300E	01C1	ABC1	BSET	D0, D1
003010	08D90004		BSET	#4, (A1)+
003014	0181		BCLR	D0, D1
003016	08920007		BCLR	#7, (A2)
00301A	0879000700003026	ABC2	BCHG	#7, STATUS
003022	0142		BCHG	D0, D2
003024	4E4D		TRAP	#13
003026	00	STATUS	DC.B	0
			END	

0142Hが**BCHG　D0, D2**のマシン語となります．また，**TRAP　#13**のマシン語は，**4E4DH**となり，以上をまとめると，**表3.15**のようなハンド・アセンブルのリストを得ることができます．

第4章

分岐命令

68000の分岐命令は，大きく無条件分岐命令と条件付分岐命令とに分類されます．無条件分岐命令には**JMP**（ジャンプ）命令，**BRA**（ブランチ）命令があり，条件付分岐命令には**Bcc**（条件付ブランチ）命令，**DBcc**（条件付デクリメント・ブランチ）命令があります．

無条件分岐命令は，無条件でオペランドで指定した先へジャンプあるいはブランチするもので，オペランドに何を記述するかによって，**JMP**（ジャンプ）と**BRA**（ブランチ）の2とおりのニーモニックが用意されています．

条件分岐命令は，ブランチ条件が成立するか否かで，オペランド先へブランチしたり，次の命令を実行したりする命令で，単なる条件付ブランチ命令（このニーモニックは**Bcc**）とカウント・レジスタのデクリメントを伴う条件付デクリメント・ブランチ命令（ニーモニックは**DBcc**）とが用意されています．

条件，無条件の両方をあわせた分岐命令は以上の4つで，これらの命令を用いて，プログラムの流れで分岐が行なわれます．

4.1　JMP命令のマシン語

JMP（**JuMP**）命令は，デスティネーション・オペランドで指定した先へ無条件でジャンプする命令で，指定できるアドレッシング・モードは，次のとおりです．

① アドレス・レジスタ間接

② ディスプレースメント付アドレス・レジスタ間接

③　インデックス・ディスプレースメント付アドレス・レジスタ間接
④　アブソリュート・ショート
⑤　アブソリュート・ロング
⑥　ディスプレースメント付プログラム・カウンタ相対
⑦　インデックス・ディスプレースメント付プログラム・カウンタ相対

プリ・デクリメント・アドレス・レジスタ間接，ポスト・インクリメント・アドレス・レジスタ間接のアドレッシング・モードは，使用することはできません．**JMP**命令を用いれば，68000のアドレス空間のどこへでもジャンプすることができます．

ただし，ディスプレースメント付プログラム・カウンタ相対のアドレッシング・モードのときはDISP16の範囲，すなわちPC±32Kの範囲内へしかジャンプすることはできません．

JMP　**<EA>** ……　<EA>で表わされる実効アドレスへ無条件ジャンプする．すなわち<EA>の実効アドレス値をPCへ転送する．

コンディション・コードは**JMP**命令では一切変化しません．

マシン語フォーマットは次のとおりです．

15	14	13	12	11	10	9	8	7	6	5　　　　0
0	1	0	0	1	1	1	0	1	1	実効アドレス

実効アドレス・フィールドは，デスティネーション・オペランドのアドレッシング・モードを指定します．

4.2　BRA命令のマシン語

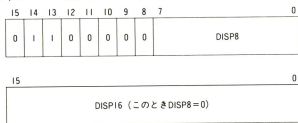
BRA（**BRAnch**）命令は，デスティネーション・オペランドで指定した飛び先へ無条件でブランチします．デスティネーション・オペランドには〈ラベル〉のみが許され，このラベル先へブランチが行なわれます．

ブランチ命令から飛び先のラベルまでの変位，ディスプレースメントには，16ビットで表現されるDISP16と，8ビットで表わされるDISP 8とがあり，これらを越える範囲でのブランチはできません．DISP 8で表現できる範囲でのブランチ命令は，1ワード長ですみ，DISP16を必要とする範囲でのブランチ命令は2ワード長が必要となります．

BRA　〈ラベル〉……　〈ラベル〉の飛び先へ無条件にブランチする．デスティネーション・オペランドには，〈ラベル〉の記述のみが許される．

X	N	Z	V	C
−	−	−	−	−

コンディション・コードは，**BRA**命令では一切変化しません．

BRA命令のマシン語フォーマットは，次のとおりです．

DISPフィールドは，**BRA**命令からラベルまでの変位，すなわちディスプレースメントの値を格納するフィールドで，8ビット表現のときはDISP 8に，また16ビット表現のときはDISP16に格納されます．DISP16が使用される場合，DISP 8のフィールドには0がセットされます．

DISP8で±128のディスプレースメントを，DISP16で±32Kのディスプレースメントを表現できます．この範囲内のラベルへしかブランチすることはできません．

4.3　Bcc命令とマシン語

Bcc（**Branch Conditionally**）命令は条件付ブランチ命令で，条件（▼cc▼で表現する）が成立すれば（すなわち真ならば）デスティネーション・オペランドで指定した〈ラベル〉へブランチし，成立しなければ（すなわち偽ならば）次の命令を実行します．

ブランチ条件は，141ページの表3.14に示すように14種類あり，ブランチ条件の2文字のニーモニックを，**Bcc**命令のccの個所へ代入して，おのおのの条件に対するブランチ命令のニーモニックができ上がります．

たとえば，ブランチ条件に▼大きい▼を取り上げると，表3.14より▼大きい▼のニーモニックは▼GT▼ですから，これを**Bcc**の▼cc▼の個所へ代入して，**BGT**命令ができ上がるという具合です．

ブランチ条件のチェックは，C（キャリー）フラグ，Z（ゼロ）フラグ，N（ネガティブ）フラグ，V（オーバーフロー）フラグの4個のフラグをチェックすることによって行なわれます．そのチェック，テストする内容の論理式と説明を，**表4.1**（次ページ）に示します．

次に**Bcc**命令を詳細に見てみましょう．

(1)　**BCC**（**Branch if Carry Clear**）

ブランチ条件：C＝0

キャリーフラグが0ならばオペランドで指定した先へブランチし，1ならば次の命令を実行します．

(2)　**BCS**（**Branch if Carry Set**）

ブランチ条件：C＝1

キャリーフラグが1ならばオペランドで指定した先へブランチし，0ならば次の命令を実行します．

(3)　**BEQ**（**Branch if EQual**）

ブランチ条件：Z＝1

ゼロ・フラグが1ならばオペランドで指定した先へブランチし，0ならば次の命令を実行します．

(4)　**BGE**（**Branch if Greater or Equal**）

ブランチ条件：（N XOR V）＝0

ネガティブ・フラグとオーバーフローフラグが等しければオペランドで指定

表4．1　ブランチ条件のチェック

ニーモニック	条　　件
CC	キャリークリア(carry clear)
CS	キャリーセット(carry set)
EQ	等しい(equal)
GE	大きい，または等しい(greater or equal)
GT	大きい(greater)
HI	高い(high)
LE	小さい，または等しい(less or equal)
LS	低い，または同じ(low or same)
LT	小さい(less)
MI	負(マイナス)(minus)
NE	等しくない(not equal)
PL	正(プラス)(plus)
VC	オーバーフロー・クリア，オーバーフローなし (overflow clear, no overflow)
VS	オーバーフロー・セット，オーバーフロー (overflow set, overflow)

した先へブランチし，等しくなければ次の命令を実行します．

(5)　BGT (Branch if GreaTer)

ブランチ条件：　((N XOR V) OR Z) = 0

ゼロ・フラグが0で，かつ，ネガティブ・フラグとオーバーフローフラグとが等しければ，すなわちN，Vともに“0”かまたは“1”であれば，オペランドで指定した先へブランチし，ゼロ・フラグが1またはネガティブ・フラグとオーバーフローフラグが等しくなければ，ブランチは行なわれず，次の命令が実行されます．

(6)　BHI (Branch if HIgh)

ブランチ条件：　(C OR Z) = 0

キャリーフラグとゼロ・フラグがともに0ならば，オペランドで指定した先

（条件の成立，不成立）

論　理　式	説　　明
$\overline{C}=1$	キャリーフラグC＝0で成立，C＝1で不成立
$C=1$	キャリーフラグC＝1で成立，C＝0で不成立
$Z=1$	ゼロ・フラグZ＝1で成立，Z＝0で不成立．
$N \cdot V+\overline{N} \cdot \overline{V}=1$	ネガティブ・フラグNとオーバーフローフラグVが等しければ（すなわちN＝V）成立，等しくなければ（N≠V）不成立．
$(N \cdot V+\overline{N} \cdot \overline{V}) \cdot \overline{Z}=1$	ゼロ・フラグZ＝0でかつネガティブ・フラグNとオーバーフローフラグVとが等しければ（すなわちN，Vともに“0”かまたは“1”）成立，Z＝1またはN≠Vならば不成立．
$\overline{C} \cdot \overline{Z}=1$	キャリーフラグCとゼロ・フラグZがともに0なら成立，C＝1またはZ＝1なら不成立．
$N \cdot \overline{V}+\overline{N} \cdot V+Z=1$	ゼロ・フラグZが1にセットされるか，またはネガティブ・フラグとオーバーフローフラグが等しくなければ（すなわちZ＝1またはN≠V）成立，ゼロ・フラグZ＝0でかつN＝Vならば不成立．
$C+Z=1$	キャリーフラグC＝1またはゼロ・フラグZ＝1ならば成立，C，Zともに0ならば不成立．
$N \cdot \overline{V}+\overline{N} \cdot V=1$	ネガティブ・フラグNとオーバーフローフラグVが等しくなければ，すなわちN≠Vならば成立，N＝Vならば不成立．
$N=1$	ネガティブ・フラグN＝1ならば成立，N＝0ならば不成立．
$\overline{Z}=1$	ゼロ・フラグZ＝0で成立，Z＝1で不成立．
$\overline{N}=1$	ネガティブ・フラグN＝0ならば成立，N＝1ならば不成立．
$\overline{V}=1$	オーバーフローフラグV＝0ならば成立，V＝1ならば不成立．
$V=1$	オーバーフローフラグV＝1ならば成立，V＝0ならば不成立．

へブランチし，キャリーフラグが1またはゼロ・フラグが1ならば，次の命令を実行します．

(7) **BLE (Branch if Less or Equal)**

ブランチ条件：((N XOR V) OR Z) = 1

ゼロ・フラグがセットされるか，またはネガティブ・フラグとオーバーフローフラグが等しくなければ，オペランドで指定した先へブランチし，ゼロ・フラグが0で，ネガティブ・フラグとオーバーフローフラグが等しければ，次の命令を実行します．

(8) **BLS (Branch if Low or Same)**

ブランチ条件：(C OR Z) = 1

キャリーフラグが1またはゼロ・フラグが1ならば，オペランドで指定した

先へブランチし，キャリーフラグ，ゼロ・フラグともに0のときは，次の命令を実行します．

(9) **BLT (Branch if Less)**

ブランチ条件：(N XOR V)＝1

ネガティブ・フラグとオーバーフローフラグが等しくなければ，指定した先へブランチし，等しければ次の命令を実行します．

(10) **BMI (Branch if MInus)**

ブランチ条件：N＝1

ネガティブ・フラグが1ならばオペランドで指定した先へブランチし，0ならば次の命令を実行します．

(11) **BNE (Branch if Not Equal)**

ブランチ条件：Z＝0

ゼロ・フラグが0ならばオペランドで指定した先へブランチし，1ならば次の命令を実行します．

(12) **BPL (Branch if PLus)**

ブランチ条件：N＝0

ネガティブ・フラグが0ならばオペランドで指定した先へブランチし，1ならば次の命令を実行します．

(13) **BVC (Branch if oVerflow Clear)**

ブランチ条件：V＝0

オーバーフローフラグが0ならばオペランドで指定した先へブランチし，1ならば次の命令を実行します．

(14) **BVS (Branch if oVerflow Set)**

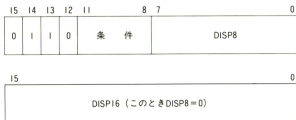
ブランチ条件：V＝1

オーバーフローフラグが1ならばオペランドで指定した先へブランチし，0ならば次の命令を実行します．

Bcc 〈ラベル〉……条件ccが成立すれば〈ラベル〉へブランチし，成立しなければ次の命令を実行する．オペランドには〈ラベル〉が記述される．

X	N	Z	V	C
−	−	−	−	−

コンディション・コードは，Bcc命令では一切変化しません．

Bcc (**BCC**, **BCS**, **BEQ**, **BGE**, **BGT**, **BHI**, **BLE**, **BLS**, **BLT**, **BMI**, **BNE**, **BPL**, **BVC**, **BVS**) 命令のマシン語フォーマットは，次のとおりです．

DISPフィールドは，BRA命令のときと同様に，Bcc命令からラベルまでの変位，すなわちディスプレースメント（Displacement）の値を格納するフィールドで，8ビットで表現する場合はDISP 8に，また16ビットで表現する場合はDISP16に格納されます．DISP16のとき，DISP 8には0がセットされます．

条件フィールドには，表3.14よりブランチ条件に対応する4ビットのコードがセットされます．BRA命令のときと同様に，DISP 8の8ビットで±128のディスプレースメントを，またDISP16の16ビットで±32Kのディスプレースメントを表現することができ，この範囲内のラベルへの条件ブランチが可能となります．

4.4　DBcc命令とマシン語

DBcc (**Decrement counter and Branch until Condition true or Count**＝－1) 命令は，カウント・レジスタのデクリメントを伴う条件付デクリメント・ブランチ命令で，条件（▼cc▼で表現する）をチェックして，これが成立すれば次の命令を実行し，もし成立しなければカウンタ・データ・レジスタを－1だけデクリメントし，その結果，データ・レジスタの値が－1になったら次の命令を実行し，そうでなければ（≠－1）オペランドで指定したところへブランチします．DBcc命令の動きを図4.1(次ページ)に示します．

DBcc命令において，条件の▼cc▼は，Bcc命令の▼cc▼(表3.14)にF(Fault, never true) とT (always true) が加わります．DBccの条件ccは，全部で16とおりあることになります．コードはおのおの次のようになります．

DBcc命令

条件は成立するか？

Y

N

(Dn)←(Dn)－１

(Dn)＝－１か？

Y

N

次の命令へ行く

オペランドで指定した先へブランチする

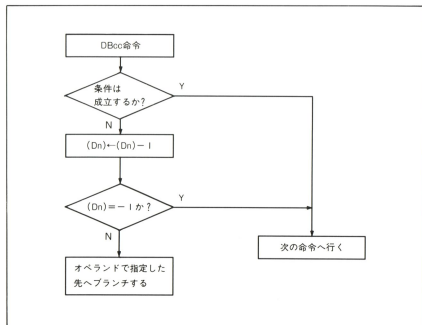



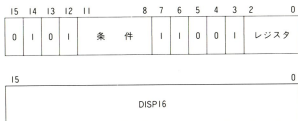
図４．１　DBcc命令の動き

ニーモニック	条件	コード
F	常に偽，常に成立せず(never true)	0 0 0 1
T	常に真，常に成立(always true)	0 0 0 0

また，カウンタ・データ・レジスタは16ビットのワードで動作しますから，デクリメントの動作はワード・オペレーションとなります．

DBcc　Dn，〈ラベル〉……条件ccが成立すれば次の命令を実行する．条件ccが成立しなければ，Dnを－１だけデクリメントして，その結果Dnの内容が－１になれば(Dn＝－１)，次の命令を実行し，－１でなければ(Dn≠－１))〈ラベル〉へブランチする．

X	N	Z	V	C
－	－	－	－	－

コンディション・コードは，DBcc命令では一切変化しません．**DBcc(DBCC, DBCS, DBEQ, DBF, (DBRA), DBGE, DBGT, DBHI, DBLE, DBLS, DBLT, DBMI, DBNE, DBPL, DBT, DBVC, DBVS)** 命令のマシン語フォーマットは次のとおりです．

条件フィールドには16とおりの条件に対応するコードがセットされます．また，レジスタ・フィールドには，カウンタ・データ・レジスタのNo.がセットされ，DISPフィールドには**DBcc**命令からラベルまでの変位，すなわちディスプレースメント値が格納されます．

次に，分岐命令のマシン語プログラミング，マシン語変換の例題を行なってみましょう．

4.5　分岐命令のマシン語プログラミング例

例題19

次のプログラムのマシン語を作りなさい．

```
        ORG     $3000
        BCS.S   ABC2
        BGE     ABC1
        BRA     XYZ1
XYZ1    DBEQ    D1, ABC2
ABC1    JMP     (A0)
ABC2    JMP     3(A1, D2)
        END
```

解き方

この例題のプログラムをマシン語に変換するのに最初に行なうことは，アセンブラのパス1の仕事，すなわち，ラベルのアドレス計算です．XYZ 1，ABC 1，ABC 2と3個のラベルがありますが，これらのアドレスを求めるには，おのおののラベルまでの全命令の長さ(Length)をまず計算しなくてはなりません．

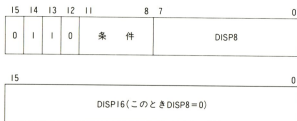
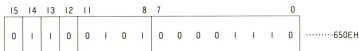
BCS.S　ABC2の命令は，ショート・タイプ(S)の指定がしてありますから，**Bcc**命令のマシン語フォーマットにおいて，DISP 8を用いてマシン語を作ればよく，2バイト長となります．

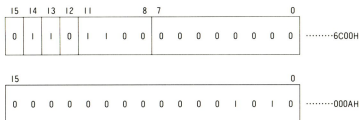
BGE　ABC1の命令は，同じマシン語フォーマットでDISP16を用いますから，4バイト長のマシン語となります．また，**BRA　XYZ1**の命令は**BRA**命令のマシン語フォーマットでDISP16を用いますから，これも4バイト長のマシン語です．さらに，**DBEQ　D1,ABC2**の命令は4バイト長のマシン語で，**JMP (A0)**の命令は，**JMP**命令のマシン語フォーマットより2バイト長となります．

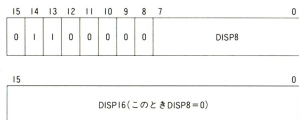
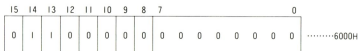
以上から，ラベル**XYZ1,ABC1,ABC2**のアドレスは，

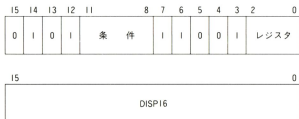
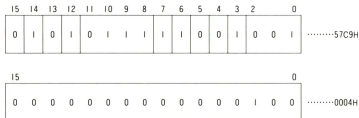
XYZ1……300AH
ABC1……300EH
ABC2……3010H

となります．

BCS.S　ABC2のマシン語は，**Bcc**命令のマシン語フォーマット；

において，条件=CS（キャリーセット）=▼0101▼（表3.14より），DISP 8 =（ABC2のアドレス−(BCS.S命令の先頭アドレス)−2）=3010H−3000H−2 = 0EHを代入して，

650EHが**BCS.S　ABC2**のマシン語となります．

BGE　ABC1のマシン語は，上と同じ**Bcc**のマシン語フォーマットに，条件＝GE（大きい，または等しい（greater or equal)＝▼1100▼（表3.14より）を代入して，またショート指定の.Sがありませんから，DISP16が用いられて，DISP16＝(ABC1のアドレス－**BGE**命令の先頭のアドレス－２)＝300EH－3002H－２＝000AH，DISP８はDISP16が用いられるときは０.これらを代入して，

6C00000AHが**BGE　ABC1**のマシン語となります．

BRA　XYZ1のマシン語は，**BRA**命令のマシン語フォーマット；

において，DISP16＝(XYZ1のアドレス－**BRA**命令の先頭アドレス－２)＝3008H－3006H－２＝0002H，DISP８＝０を代入して，

60000002Hが**BRA　XYZ1**のマシン語となります．

XYZ1　DBEQ　D1, ABC2のマシン語は，**DBcc**命令のマシン語フォーマット；

において，条件＝EQ（等しい：equal）＝▼0111▼（表3.14より），レジスタ・フィールド＝D1使用＝▼001▼，DISP16＝（**ABC2**のアドレス－**DBEQ**命令の先頭アドレス－2）＝3010H－300AH－2＝0004Hを代入して，

57C90004Hが**XYZ1　DBEQ　D1, ABC2**のマシン語となります．

ABC1　JMP（A0）のマシン語は，**JMP**命令のマシン語フォーマット；

15	14	13	12	11	10	9	8	7	6	5　　　　0
0	1	0	0	1	1	1	0	1	1	実効アドレス

において，実効アドレス＝A０を用いたアドレス・レジスタ間接のアドレッシング・モード＝▼010000▼を代入して，

15	14	13	12	11	10	9	8	7	6	5					0	
0	1	0	0	1	1	1	0	1	1	0	1	0	0	0	0	………4ED0H

4ED0Hが**ABC1　JMP　(A0)**のマシン語となります．

ABC2　JMP　3(A1,D2)のマシン語は，上と同じマシン語フォーマットに，実効アドレス・フィールド＝インデックス・ディスプレースメント付アドレス・レジスタ間接アドレッシング・モード＝▼110001▼を代入して，

（001の部分：A1）

15	14	13	12	11	10	9	8	7	6	5					0	
0	1	0	0	1	1	1	0	1	1	1	1	0	0	0	1	………4EF1H

4EF1Hが**ABC2　JMP　3（A1,D2）**のマシン語の第１ワードとなります．

インデックス・ディスプレースメント付アドレス・レジスタ間接モードでは，さらにインデックス・レジスタを指定するための拡張ワードが１ワード必要となり，そのフォーマットは次のとおりです．

15	14　　12	11	10	9	8	7　　0
D/A	レジスタ	W/L	0	0	0	DISP8

D／A…インデックス・レジスタがデータ・レジスタかアドレス・レジスタかを指定するビットで，０でデータ・レジスタを，１でアドレス・レジスタを指定する．

レジスタ…インデックス・レジスタのNo.をセットする．

W／L…０でインデックス・レジスタの下位ワード（16ビット）の値が符号

拡張される．

1でインデックス・レジスタのロング・ワード（32ビット）がそのまま使われる．

DISP 8…8ビットのディスプレースメント値がセットされる．

これにD／Aフィールド＝インデックス・レジスタはデータ・レジスタ＝0，レジスタ・フィールド＝インデックス・レジスタNo.(D 2)＝▼010▼，W／Lフィールド＝ワード＝▼0▼，DISP 8＝3を代入して，

15	14	12	11	10	9	8	7							0	
0	0 1	0	0	0	0	0	0	0	0	0	0	0	1	1	………2003H

2003Hが拡張ワードとなり，以上から**ABC2　JMP　3(A1,D2)**のマシン語は，

4EF12003H

となります．

これらをまとめると**表4.2**のようなハンド・アセンブルのリストを得ることができます．

表4.2　例題19のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	650E		BCS.S	ABC2
003002	6C00000A		BGE	ABC1
003006	60000002		BRA	XYZ1
00300A	57C90004	XYZ1	DBEQ	D1, ABC2
00300E	4ED0	ABC1	JMP	(A0)
003010	4EF12003	ABC2	JMP	3(A1, D2)
			END	

第5章

サブルーチンの呼び出し，リターン命令

サブルーチン呼び出しにはJSR(ジャンプ・サブルーチン)，BSR(ブランチ・サブルーチン）命令があり，サブルーチンから戻るにはRTS（リターン・サブルーチン），RTR（リターン・サブルーチン，復元CC）命令があり，これらの命令を使ってサブルーチンへの呼び出し，リターンを行ないます．この関係を図5.1に示します．

オペランドで指定したサブルーチンへ飛ぶには，ジャンプ（JSR）とブランチ（BSR）の2とおりの命令が用意されており，これらのどちらかを用いてサブルーチンの呼び出しを行ないます．

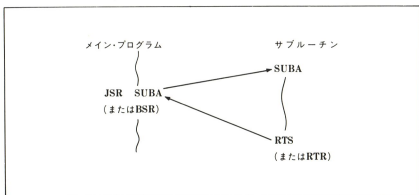



図5.1　サブルーチンの呼び出しと戻り

5.1　JSR命令とマシン語

JSR (Jump to SubRoutine) 命令は，JSR命令の次の命令（サブルーチンから戻って最初に実行する命令）のアドレスをシステム・スタックに格納し，オペランドで指定した先へジャンプします．JSR命令のオペランドで指定できるアドレッシング・モードは，前章のJMP命令とまったく同じ要領ですが，もう一度それらを次に示します．

① アドレス・レジスタ間接
② ディスプレースメント付アドレス・レジスタ間接
③ インデックス・ディスプレースメント付アドレス・レジスタ間接
④ アブソリュート・ショート
⑤ アブソリュート・ロング
⑥ ディスプレースメント付プログラム・カウンタ相対
⑦ インデックス・ディスプレースメント付プログラム・カウンタ相対

（プリ・デクリメント・アドレス・レジスタ間接，ポスト・インクリメント・アドレス・レジスタ間接のアドレッシング・モードは使用不可）

したがって，JMP命令を用いれば68000のアドレス空間内のどこへでもジャンプすることができたように，JSR命令を用いてサブルーチンを呼び出せば，そのサブルーチンはアドレス空間内のどこにでも好きなところに置くことができます．後述するBSR命令とこのJSR命令との違いは，JMP命令とBRA（ブランチ）命令との違いであるともいうことができます．

JSR命令はサブルーチンへ飛んで行くのにJMP（ジャンプ）命令を用い，BSR命令はBRA（ブランチ）命令を用いてサブルーチンへ飛んでいきます．BRA（ブランチ）命令で飛べる範囲は，最大でDISP16で表現可能な範囲ですから±32K，したがって，この範囲内にあるサブルーチンの呼び出しにはBSR命令が使えますが，それを超えるアドレスにあるものに対しては，JSR命令を使う以外に，そのサブルーチンを呼び出すことはできません．

JSR　〈EA〉……プログラム・カウンタの値，すなわちこの命令の次の命令のアドレスをシステム・スタックに格納し，〈EA〉で表わされる実効アドレスへ無条件ジャンプする．つまり〈EA〉の実効アドレス値をPC（プログラム・カウンタ）へ転送する．

コンディション・コードは，JSR命令ではいっさい変化しません．
マシン語フォーマットは次のとおりです．

15	14	13	12	11	10	9	8	7	6	5　　　　0
0	1	0	0	1	1	1	0	1	0	実効アドレス

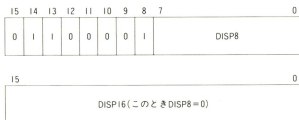
実効アドレス・フィールドは，デスティネーション・オペランドのアドレッシング・モードを指定します．

5.2　BSR命令とマシン語

BSR (Branch to SubRoutine) 命令は，BSR命令の次の命令（サブルーチンから戻って最初に実行する命令）のアドレスをシステム・スタックに格納し，オペランドで指定したラベルへブランチします．BSR命令から飛び先のラベル（サブルーチン名）までの変位，ディスプレースメントには16ビットで表現されるDISP16と，8ビットで表現されるDISP8とがあって，この範囲内でしかサブルーチンにブランチすることはできません．これを超える場合には，前述したようにJSR命令を用いることになります．

BSR 〈ラベル〉……プログラム・カウンタの値，すなわちこの命令の次の命令のアドレスをシステム・スタックに格納し，〈ラベル〉で表わされる飛び先へ無条件にブランチする．デスティネーション・オペランドには〈ラベル〉を指定する．

コンディション・コードは，BSR命令ではいっさい変化しません．

マシン語フォーマットは次のとおりです．

DISPフィールドは，BSR命令からラベルまでの変位すなわちディスプレースメントの値を格納するフィールドで，8ビット表現のときはDISP 8に，また16ビット表現のときはDISP16に格納されます．DISP16が使用される場合，DISP 8フィールドには0（ゼロ）がセットされます．

DISP 8で±128のディスプレースメントを，またDISP16で±32Kのディスプレースメントを表現でき，前述したように，この範囲内にあるサブルーチンのラベルへ**BSR**命令を使って飛ぶことはできますが，範囲外の場合は**BSR**命令では無理で，**JSR**命令を使うことになります．

5.3　RTS命令とマシン語

RTS（ReTurn from Subroutine）命令は，サブルーチンからメイン・プログラムに戻るための命令で，システム・スタック中に格納されている戻りアドレスをプログラム・カウンタに持ってきて，そのアドレスから実行する命令です．

RTS……システム・スタックから戻りアドレスを持ってきて，これをプログラム・カウンタ（PC）に転送し，そのアドレスから実行を開始する．

X	N	Z	V	C
−	−	−	−	−

コンディション・コードは，RTS命令ではいっさい変化しません．

マシン語は次のとおりです．

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

5.4 RTR命令とマシン語

RTR(**ReTurn from subroutine and Restore CC**)命令は，サブルーチンからメイン・プログラムへ戻るための命令ですが，**RTS**命令が単に戻るだけだったのに対し，この**RTR**命令はコンディション・コードをもスタックから取り出し，これをコンディション・コード・レジスタに復元して戻ります．このとき，ステータス・レジスタSRの下位8ビットのユーザーバイトのみが変化し，上位8ビットのシステム・バイトは影響を受けません．

RTR……システム・スタックからコンディション・コードを持ってきて，これをコンディション・コード・レジスタへ復元し，ついでシステム・スタックから戻りアドレスを持ってきて，これをプログラム・カウンタ(PC)に転送し，そのアドレスから実行を開始する．SRの上位8ビット（システム・バイト）は変化しません．

X	N	Z	V	C
*	*	*	*	*

コンディション・コードはすべて変化し，システム・スタックから取り出した内容がセットされます．

マシン語は次のとおりです．

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

なお，RTR命令を用いてサブルーチンからリターンする場合は，当然のことながら，サブルーチンの先頭でコンディション・コードをスタックに退避しておいてやらなくてはなりません．

5.5 サブルーチンの呼び出し，リターン命令のマシン語プログラミング例

例題20　次のプログラムをマシン語に変換しなさい．

```
        ORG       $3000
        PEA       PARA1
        PEA       PARA2
        BSR       SUBA
        TRAP      #13
PARA1   DC.L      1
PARA2   DC.L      10
SUBA    MOVEA.L   4(SP), A0
        MOVEA.L   8(SP), A1
        MOVE.L    (A0), D0
        MOVE.L    (A1), D1
        ADD.L     D1, D0
        ADD.W     $300, D0
        MOVE.L    D0, 4(SP)
        RTS
        END
```

解き方
このプログラム例は，メイン・ルーチンとサブルーチンとの間でのパラメータ，結果のやり取りにスタック領域を使う例です．まず最初に，メイン・ルーチンでは，サブルーチンを呼ぶ前にサブルーチンに渡すべきパラメータをスタックにセットしておきます．サブルーチンへ渡すのに，パラメータそのものを

渡す方法と，パラメータの格納されているアドレスを渡す方法とがありますが，ここでは後者のパラメータのアドレスを渡すやり方でプログラムしてあります，

メイン・ルーチンからサブルーチンへ渡すべきパラメータは，PARA1とPARA2で，これらのパラメータのアドレスが，

PEA　PARA1

PEA　PARA2

命令によって，スタック領域にプッシュされます．

次に，

BSR　SUBA

命令によって，サブルーチンSUBAをコールし，SUBAに制御が渡ります．この時点で，スタックにはPARA1のアドレス，PARA2のアドレス，そしてスタックのトップ（Top Of Stack＝TOS）には戻りアドレス（サブルーチンからメインルーチンへの戻りアドレス）という順序で格納されています．

サブルーチンでは最初に，メイン・ルーチンから渡されたパラメータのアドレスをスタックから取り出します．

これは，

MOVEA.L　4(SP)，A0

MOVEA.L　8(SP)，A1

命令によって行なわれます．

これらを用いて，サブルーチンでの内部処理が終わったら，結果をメイン・ルーチンに渡さなくてはなりませんが，ここで再びスタック受渡しを用います．

D0レジスタにサブルーチンの内部処理の結果が入っており，これを，

MOVE.L　D0，4(SP)

命令でスタックに格納して，最後にRTS命令でサブルーチンからメイン・ルーチンに戻ります．

メイン・ルーチンでは，その結果をスタックから取り出して，次の処理へ移っていきます．このようなスタック域を用いたパラメータの受渡しは，通常もっともよく用いられる受渡し方法です．

まず，PEA　PARA1命令をマシン語に変換します．PEA命令のマシン語フォーマットは，

15	14	13	12	11	10	9	8	7	6	5 ～ 0
0	1	0	0	1	0	0	0	0	1	実効アドレス

で，実効アドレス・フィールド＝アブソリュート・ロング＝▼111001▼を代入して，

15	14	13	12	11	10	9	8	7	6	5					0	
0	1	0	0	1	0	0	0	0	1	1	1	1	0	0	1	………4879H

4879Hがマシン語の第1ワードとなります．

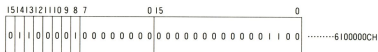
次に，**PARA1**のアドレス▼00003012H▼が続いて，**PEA　PARA1**のマシン語は，**487900003012H**となります．

PEA　PARA2命令は，同様に**487900003016H**となります．

また，サブルーチンをコールする**BSR　SUBA**の命令は，**BSR**のマシン語フォーマット；

15	14	13	12	11	10	9	8	7　　　0	15　　　0
0	1	1	0	0	0	0	1	DISP8	DISP16(このときDISP8＝0)

に，DISP16＝命令とラベルとのディスプレースメント＝(**SUBA**のアドレス)－(**BSR**命令のアドレス)－２＝301AH－300CH－２＝CHを代入して，

15	14	13	12	11	10	9	8	7　　　0	15　　　0	
0	1	1	0	0	0	0	1	00000000	0000000000001100	………6100000CH

6100000CHが，**BSR　SUBA**のマシン語となります．

TRAPのマシン語フォーマットは，

15	14	13	12	11	10	9	8	7	6	5	4	3　　　0
0	1	0	0	1	1	1	0	0	1	0	0	トラップ・ベクタ番号

で，トラップ・ベクタ番号フィールドに#13を代入して，**TRAP　#13**のマシン語は，

4E4DH

となります．

MOVEA.L 4(SP)，A0のマシン語は，**MOVEA**のマシン語フォーマット；

15	14	13 12	11 9	8 6	5 3	2 0
0	0	サイズ	デスティネーション (レジスタ)	0 0 1	ソース (モード)	(レジスタ)

に，サイズ・フィールド＝ロング・ワード・オペレーション＝▼10▼，デスティネーション・レジスタ・フィールド＝A０のレジスタ番号＝▼000▼をまず代入します．

ソース・フィールドで第１（ソース）オペランドのアドレッシング・モードを指定します．第１オペランドは，

4(SP)

で，このアドレッシング・モードは“ディスプレースメント付アドレス・レジスタ間接”で，アドレス・レジスタ番号＝７＝▼111▼ですから，ソース・フィールド＝▼101111▼となります．以上から，**MOVEA.L 4(SP)，A0**のマシン語の第１ワードは，

15	14	13 12	11 9	8 6	5 3	2 0	
0	0	1 0	0 0 0	0 0 1	1 0 1	1 1 1	………206FH

206FHとなります．続いて，ディスプレースメントのワード**0004H**がきますから，

206F0004H

が**MOVEA.L 4(SP)，A0**のマシン語となります．また，**MOVEA.L 8(SP)，A1**のマシン語は同様にして，

226F0008H

となります．

MOVE.L (A0)，D0のマシン語は，

15	14	13 12	11 9	8 6	5 3	2 0
0	0	サイズ	デスティネーション (レジスタ)	(モード)	ソース (モード)	(レジスタ)

に，サイズ・フィールド＝ロング・オペレーション＝▼10▼，デスティネーション・フィールド＝データ・レジスタ番号＝D０＝直接アドレッシング・モード＝

▼000000▼，ソース・フィールド＝(A０)＝アドレス・レジスタ間接アドレッシング・モード＝▼010000▼を代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	………2010H

2010Hとなります．

MOVE.L　(A1)，D1のマシン語も同様にして，**2211H**となります．

ADD.L　D1,D0のマシン語は，**ADD**のマシン語フォーマット；

15	14	13	12	11 ― 9	8 ― 6	5 ― 0
1	1	0	1	レジスタ	OPモード	実効アドレス

に，レジスタ・フィールド＝データ・レジスタ番号＝D０の０（ゼロ）＝▼000▼，オペレーション・モード・フィールド＝ロング・ワード・オペレーション＝▼010▼，実効アドレス・フィールド＝データ・レジスタ（D１）直接アドレッシング・モード＝▼000001▼を代入して，

15	14	13	12	11		9	8		6	5					0	
1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	1	………D081H

D081Hが**ADD.L　D1,D0**のマシン語となります．

ADD.W　$300,D0のマシン語は，同じ**ADD**のマシン語フォーマットに，レジスタ・フィールド＝▼000▼，オペレーション・モード・フィールド＝ワード・オペレーション＝▼001▼，実効アドレス・フィールド＝▼111000▼を代入して，

15	14	13	12	11		9	8		6	5					0	
1	1	0	1	0	0	0	0	0	1	1	1	1	0	0	0	………D078H

D078Hとなりますが，これはマシン語の第１ワードで，この後に$300＝0300Hが続きます．したがって，**ADD.W　$300,D0**のマシン語は，**D0780300H**となります．

MOVE.L　D0, 4(SP)のマシン語は，**MOVE**命令のマシン語フォーマット

に，サイズ・フィールド＝ロング・ワード・オペレーション＝▼10▼，デスティネーション・フィールド＝ディスプレースメント付アドレス・レジスタ間接アドレッシング・モード〔4 (SP)〕＝▼111101▼，ソース・フィールド＝データ・レジスタ（D0）直接アドレッシング・モード＝▼000000▼を代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	1	1	1	1	0	1	0	0	0	0	0	0	………2F40H

2F40Hが，**MOVE.L　D0,4(SP)**のマシン語の第1ワードで，これにディスプレースメント**0004H**が続いて，

2F400004H

がマシン語となります．

最後に，**RTS**命令でサブルーチンからメイン・ルーチンへ戻りますが，**RTS**のマシン語は，

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1	………4E75H

4E75Hです．以上をまとめると，次のページの**表5.1**のようなハンド・アセンブル・リストが得られます．

表5．1　例題20のハンド・アセンブル・リスト

	00003000		ORG	$3000
003000	487900003012		PEA	PARA1
003006	487900003016		PEA	PARA2
00300C	6100000C		BSR	SUBA
003010	4E4D		TRAP	#13
003012	00000001	PARA1	DC.L	1
003016	0000000A	PARA2	DC.L	10
00301A	206F0004	SUBA	MOVEA.L	4(SP), A0
00301E	226F0008		MOVEA.L	8(SP), A1
003022	2010		MOVE.L	(A0), D0
003024	2211		MOVE.L	(A1), D1
003026	D081		ADD.L	D1, D0
003028	D0780300		ADD.W	$300, D0
00302C	2F400004		MOVE. L	D0, 4(SP)
003030	4E75		RTS	
			END	

第6章

LINK,UNLK命令

サブルーチンやプロシージャの先頭で，スタックにローカル変数領域を確保し，戻る際には，確保した領域を解放するオペレーションが必要ですが，これは，サブルーチン，プロシージャのコーディングで必ず出てくる決まりきった事柄です．これで自己再帰（セルフ・リカーシブ），再入可能（リエントラント）をサポートすることができます．このために68000に設けられた命令が，LINK命令とUNLK命令です．LINK命令は領域を確保し，UNLK命令はこの確保した領域を解放します．

それでは，少し話はとびますが，このLINK，UNLKというような命令が用意されていないと，自己再帰とか再入可能なプログラムは組めないのでしょうか．答はもちろん“否”です．こうした命令が別に用意されていなくても，その命令内容を別の命令を使って作ってやればよいだけの話です（ステップ数は増えますが）．その例として，16ビット・マイコン8086でのLINK，UNLK処理を考えてみましょう．

6.1 8086におけるLINK，UNLK処理

8086にはLINK，UNLKという命令はサポートされていませんから，LINK処理，UNLK処理を行なうには他の命令を使って，同じ処理内容を実現しなくてはなりません．

リンク処理は，

① ベース・レジスタの内容をスタックに退避するためプッシュする．

②　スタック・ポインタ内容をベース・レジスタに転送する．
③　スタック・ポインタを確保する領域のバイト数分だけ減じる．
8086では，スタック領域のベース・レジスタとしてBPを使いますから，

①　**PUSH　BP**
②　**MOV　BP, SP (SP→BP)**
③　**ADD　SP, −4** (確保領域4バイト)

の3ステップを実行してやれば，リンク処理が行なわれたことと等価となります．

アンリンク処理は，

①　スタック・ポインタにベース・レジスタの値を転送する．
②　ベース・レジスタに，退避しておいた値を復元する．

①　**MOV　SP,BP (BP→SP)**
②　**POP　BP**

以上のリンク処理は3命令，アンリンク処理は2命令を要したわけですが，これを1命令で実行してしまうように作られたものが，68000の**LINK**命令と**UNLK**命令というわけです．

6.2　LINK命令

LINK (LINK and allocate) 命令は，スタックに領域を確保します．ベース・レジスタとして，第1オペランドで指定したアドレス・レジスタの内容をスタックにプッシュし，次にこのスタック・ポインタSPの値を同じアドレス・レジスタに転送し，最後にスタック・ポインタに第2オペランドで指定した〈ディスプレースメント〉の値を加算して，**LINK**命令の処理を終了します．

簡単にいえば，ベース・ポインタとして使用するアドレス・レジスタ内容をスタックに退避し，これにベース・アドレス(その時点でのSP値)をセットし，SPを確保したい領域のバイト数分だけ減じる一連のオペレーションが，**LINK**命令の中身です．

アセンブラ記述は，第1オペランドにベース・アドレスを保持するベース・レジスタとして用いるアドレス・レジスタを指定し，第2オペランドには確保したい領域のバイト数が入るディスプレースメントを指定します．

LINK An，#〈ディスプレースメント〉

………Anをスタックにプッシュし，SPの内容をAnに転送し，SPにディスプレースメント値を加える．

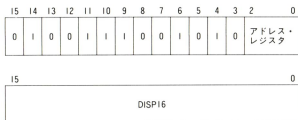
An→－(SP)

SP→An

SP＋DISP→SP

X	N	Z	V	C
－	－	－	－	－

コンディション・コードは，LINK命令 はいっさい変化しません．

マシン語フォーマットは次のとおりです．

アドレス・レジスタ・フィールドはベース・レジスタとして使用するアドレス・レジスタの番号を指定し，DISP16フィールドは確保したい領域のバイト数を負にして（2の補数）指定します．

6.3 UNLK命令

UNLK（UNLinK）命令は，LINK命令と対で用いられ，LINK命令でスタック上に確保した領域を解放するのに用います．UNLK命令では，オペランドで指定したアドレス・レジスタの内容をSPに転送し，スタックに退避してあった値を，このアドレス・レジスタに復元します．

したがって，当然のこととして，LINK命令で指定したアドレス・レジスタがUNLK命令でも指定されることになります．LINK命令とUNLK命令で別のア

ドレス・レジスタが指定されると，退避されたレジスタの値が別のレジスタに復元されてしまうことになりますから，同一のアドレス・レジスタが指定されなくてはなりません．

このUNLK命令の処理の最初の部分，すなわちアドレス・レジスタの内容をSPに転送するのは，これによってSPがベース・レジスタと同じところをポイントするようにするためで，LINK命令で確保した領域のバイト数(ディスプレースメント値）分だけ，（あるいは領域を確保した上でさらにプッシュが行なわれておればその分も含めて）元に戻る（解放する）ことになります．したがって，サブルーチンの内部では，このベース・レジスタの値を変えないことが前提となります．このベース・レジスタとして用いられるアドレス・レジスタのいろいろなアドレッシング・モード，たとえばディスプレースメント付アドレス・レジスタ間接モードやインデックス・ディスプレースメント付アドレス・レジスタ間接モードを使って，LINK命令で確保した領域にアクセスすることができます．

UNLK　An……　Anの内容をSPに転送し，スタックに退避しておいた値をAnにポップし，これを復元する．
An→SP
(SP)＋　→An

X	N	Z	V	C
−	−	−	−	−

コンディション・コードは，LINK命令ではいっさい変化しません．

マシン語フォーマットは次のとおりです．

15	14	13	12	11	10	9	8	7	6	5	4	3	2　　0
0	1	0	0	1	1	1	0	0	1	0	1	1	アドレス・レジスタ

アドレス・レジスタ・フィールドでアドレス・レジスタの番号を指定します．

6.4　LINK，UNLK命令の使い方

サブルーチンSUBAでLINK，UNLK命令を用いる例を図6.1に示します．次に，スタックがどのように変化していくのかを見てみましょう（次ページ図6.2参照）．

メイン・ルーチンの，

JSR　SUBA

で戻り番地がスタックに格納されます(①)．それから，制御はサブルーチンSUBAの先頭へいき，ここで，

LINK A0，#－4

命令が実行されます．LINK命令では，まずA０の内容を退避するため，これをスタックにプッシュします．次に，プッシュ操作で－４（４バイト，１ロング・ワード）減じたSP（スタック・ポインタ）値をA０レジスタに転送し，ベース・レジスタA０の初期化をします．さらに，SPにディスプレースメント値，－４を加えて，変数領域を確保します．つまり，変数領域のバイト数分だけSP値が減じられ，そのTOP(頂上)をSPがポイントすることになります(②)．また，

MOVE.L　D0，－4（A0）

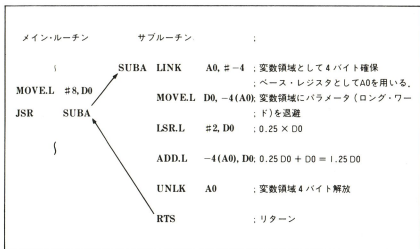



図６．１　LINK, UNLK命令の使用例

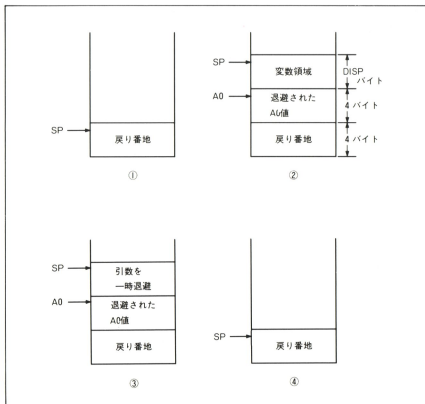



図6.2　LINK, UNLK命令とスタックの状態

命令では，レジスタD0経由で**SUBA**に渡されたパラメータ，引数を**LINK**命令で確保した領域に一時退避しておきます(③)．このとき，変数領域へのアドレッシングは，ベース・レジスタA0を用いたアドレッシング・モード（この場合は，ディスプレースメント付アドレス・レジスタ間接）で行なわれます．

LSR.L　#2, D0

命令では，D0レジスタの内容を2ビット右へ論理シフトして，0.25×D0を計算し，D0にそのまま残しておきます．次に，

ADD.L　−4 (A0)，D0

命令で，先ほど変数域に退避しておいた値とD0とを加算し，これをD0へ格納

します．これにより，0.25D0＋D0＝1.25D0が求まり，結果はD0レジスタにセットされます．

UNLK　A0

命令では，ベース・レジスタA0の値をスタック・ポインタSPへ転送して，SPをベース・レジスタA0と同じアドレスをポイントさせ，次にベース・レジスタA0に元の値を復元します(④)．そして，

RTS

命令で，スタックにある戻り番地をプログラム・カウンタに持ってきて，そのアドレスからプログラムを再開します．すなわち，サブルーチンからメイン・ルーチンに制御が戻り，メイン・ルーチンの**JSR**命令の次の命令から実行が開始されます．

6.5　LINK，UNLK命令のマシン語プログラミング例

例題21

次のプログラムをマシン語に変換しなさい．

```
        ORG     $3000
SUBA    LINK    A0, #-4
        MOVE.L  D0, -4(A0)
        LSR.L   #2, D0
        ADD.L   -4(A0), D0
        UNLK    A0
        RTS
        END
```

解き方

LINK命令のマシン語フォーマットは，

15	14	13	12	11	10	9	8	7	6	5	4	3	2　　0
0	1	0	0	1	1	1	0	0	1	0	1	0	アドレス・レジスタ

15　　　　　　　　　　0
DISP16

で，アドレス・レジスタはA０ですからアドレス・レジスタ・フィールド＝A０のレジスタ番号＝▼000▼，ディスプレースメントは－４ですからDISP16＝0FFFCHとなり，これらを代入して，

4E50FFFCHが**LINK　A0,#－4**のマシン語となります．

MOVE.L　D0,－4（A0）のマシン語は，**MOVE**命令のマシン語フォーマット：

15	14	13　12	11　　9	8　　6	5　　3	2　　0
0	0	サイズ	デスティネーション（レジスタ）	（モード）	ソース（モード）	（レジスタ）

を用いてマシン語に変換します．

サイズ＝ロング・ワード・オペレーション＝▼10▼で，ソース・フィールドはモード＝データ・レジスタ直接＝▼000▼，レジスタ＝D０＝▼000▼となり，またデスティネーション・フィールドはモード＝ディスプレースメント付アドレス・レジスタ間接＝▼101▼，レジスタ＝A０＝▼000▼となり，これらをマシン語フォーマットに代入して，

15	14	13	12	11		9	8		6	5		3	2		0	
0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	………2140H

2140Hが**MOVE.L　D0，－4(A0)**の最初のオペレーション・ワードとなります．これにディスプレースメント値，－4をデスティネーション実効アドレス・オペランド・フィールドにセットして（この値はFFFCHですから），以上をまとめて，

2140FFFCH

が**MOVE.L　D0，－4（A0）**のマシン語となります．

LSR.L　#2，D0のマシン語は，**LSR**命令のマシン語フォーマット（レジスタ内容をシフトする場合）；

15	14	13	12	11　　9	8	7　6	5	4	3	2　　0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	0	1	レジスタ

に，カウント/レジスタ・フィールド＝#2＝▼010▼，サイズ＝ロング・ワード・オペレーション＝▼10▼，i/r＝シフトするビット数を即値で指定＝0，レジスタ・フィールド＝D0＝▼000▼となり，これらを代入して，

15	14	13	12	11		9	8	7	6	5	4	3	2		0	
1	1	1	0	0	1	0	0	1	0	0	0	1	0	0	0	………E488H

E488Hが**LSR.L　#2，D0**のマシン語となります．

ADD.L　－4(A0)，D0のマシン語は，**ADD**命令のマシン語フォーマット；

15	14	13	12	11　　9	8　　6	5　　　　0
1	1	0	1	レジスタ	OPモード	実効アドレス

に，レジスタ・フィールド＝D0＝▼000▼，オペレーション・モード・フィールド＝ロング・ワードでDn＋〈EA〉→Dn＝▼010▼，実効アドレス・フィールド＝DISP付アドレス・レジスタ間接＝モード・フィールド（▼101▼）＋レジスタ・フィールド（A0＝▼000▼）＝▼101000▼，これらを代入して，

15	14	13	12	11		9	8		6	5					0	
1	1	0	1	0	0	0	0	1	0	1	0	1	0	0	0	………D0A8H

D0A8Hが第1ワードで，次にDISPフィールド＝－4＝FFFCHを加えて，

D0A8FFFCH

が**ADD.L　－4（A0），D0**のマシン語となります．

UNLK　A0のマシン語フォーマット；

15	14	13	12	11	10	9	8	7	6	5	4	3	2		0
0	1	0	0	1	1	1	0	0	1	0	1	1	アドレス・レジスタ		

に，アドレス・レジスタ・フィールド＝A0＝▼000▼を代入して，

15	14	13	12	11	10	9	8	7	6	5	4	3	2		0	
0	1	0	0	1	1	1	0	0	1	0	1	1	0	0	0	………4E58H

4E58Hが**UNLK　A0**のマシン語となります．

RTS命令のマシン語は**4E75H**となります．

以上をまとめると，**表6.1**のようなハンド・アセンブル・リストが得られます．

表6.1　例題21のハンド・アセンブル・リスト

```
          00003000             ORG      $3000
003000    4E50FFFC    SUBA     LINK     A0, #-4
003004    2140FFFC             MOVE.L   D0, -4(A0)
003008    E488                 LSR.L    #2, D0
00300A    D0A8FFFC             ADD.L    -4(A0), D0
00300E    4E58                 UNLK     A0
003010    4E75                 RTS
                               END
```

第7章

トラップ発生命令

ソフトウェアでトラップ(Trap)を発生するのに用いるのがトラップ発生命令です．スーパバイザ・コール(Supervisor Call)，オペレーティング・システム・コール(Operating System Call)，モニタ・コールなどで用いられ，実行中のユーザープログラムからシステム・プログラムへ制御が渡されます．

たとえば，リアルタイム・マルチタスク・モニタのシステム・コールを呼ぶには，

```
MOVE.L    #10,D0
MOVE.L    #20,A0
TRAP      #1
```

のように，レジスタにモニタへ渡すパラメータをセットしておき，トラップ発生命令でトラップ(Trap)を発生させ，リアルタイム・マルチタスク・モニタへ制御を渡します．そして，モニタ内では，渡されたパラメータをもとにして，それで指定されたシステム・コールの処理を実行します．

このように，トラップを発生させ，これによってCPUに“例外処理”を行なわせ，特殊な処理プログラムを実行させるのにトラップ発生命令が使われます．トラップ発生命令には，**TRAP**，**TRAPV**，**CHK**の３つの命令があります．

TRAP命令はトラップを無条件で起こす命令，**TRAPV**命令はオーバーフローフラグが▼１▼のときトラップを起こす命令，**CHK**命令は境界チェック(Check)を行ない，はみ出たらトラップを起こす命令です．

次に，これら３種類のトラップ発生命令の動作，機能，マシン語を詳しく見てみることにしましょう．

7.1 TRAP命令とマシン語

TRAP(Trap)命令は，トラップを無条件で発生させる命令で，この発生により，CPUは例外処理（Exception Processing）を開始します．TRAP命令のオペランドには，"トラップ・ベクタ番号"として，0から15までの番号を指定できます．

トラップ・ベクタ番号0～15は，例外ベクタ番号32～47におのおの対応し，この例外ベクタ番号を4倍して（すなわち2ビット左へシフトして）例外ベクタ・アドレスが求まり，このアドレスにトラップ処理プログラム，トラップ処理ルーチンの先頭アドレスをセットしておくと，トラップ発生により，制御がこのトラップ処理プログラムの先頭に渡されます．例えば，オペレーティング・システムでよく使うトラップ・ベクタ番号に"#1"があります．

パラメータをレジスタなどにセットしてから，

```
TRAP  #1
```

を実行します．

トラップ・ベクタ番号は1ですから，例外ベクタ番号（例外ベクタ番号＝トラップ・ベクタ番号＋32）は33となり，この例外ベクタ番号33を4倍して，132番地が求まり，ここにトラップ処理プログラム，すなわちOSの先頭アドレス(入口）が格納されます．こうして，

```
TRAP  #1
```

の#1トラップの発生によりOSへ制御が渡されていきます．

次に，オペレーティング・システムとしてリアルタイム・マルチタスク・モニタのRMS68Kのシステム・コールを，いくつか例にとって考えてみましょう．

(1) **DELAYシステム・コール**

このシステム・コールは，タスクを指定した実時間だけ遅延(ディレイ，Delay)して，遅延時間の経過後，そのタスクを再びレディ(Ready)状態にするもので，実時間インタバルでプログラムを走らせたりするのに使うことができます．DELAYシステム・コールを呼ぶには，#21をD0レジスタに，遅延時間(1msecの倍数)をA0レジスタにセットして，#1のトラップ命令を実行します．たとえば，100msecのDelay値を持つようにシステム・コールを発信するには，

```
MOVE.L  #21,D0
MOVE.L  #100,A0
```

TRAP　#1

とトラップ命令を使い，これによって100msecのディレイ・システム・コールが実行されます．

(2) **RESUMEシステム・コール**

このシステム・コールは，タスクの実行を再開するもので，サスペンド(Suspend)状態のタスクをレディ状態に遷移させます．**RESUME**システム・コールを呼ぶには，#18をD0レジスタに，パラメータ・ブロック・アドレスをA0レジスタにセットして，#1のトラップ命令を実行します(図7.1)．

```
            MOVE.L   #18, D0
            MOVE.L   #PRBLK, A0
            TRAP     #1
              :
PRBLK       EQU      *
            DC.L     'TAKZ'
            DC.L     0
```

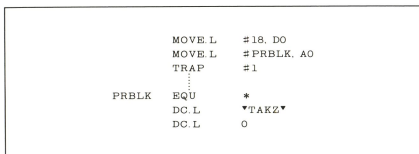



図7.1　RESOMEシステム・コール

(3) **STARTシステム・コール**

このシステム・コールにより，タスクがスタート(Start)し，開始されます．**START**システム・コールを呼ぶには，#13をD0レジスタに，パラメータ・ブロックのアドレスをA0レジスタにセットして，#1のトラップ命令を実行します(次ページ図7.2参照)．

```
                MOVE.L   #13, D0
                MOVE.L   #PRBLK, A0
                TRAP     #1
                  :
PRBLK           EQU      *
                DC.L     'TSKZ'
                DC.L     0
                DC.W     0
                DC.L     0
                DC.L     0
RD0             DC.L     0
RD1             DC.L     0
RD2             DC.L     0
RD3             DC.L     0
RD4             DC.L     0
RD5             DC.L     0
RD6             DC.L     0
RD7             DC.L     0
RA0             DC.L     0
RA1             DC.L     0
RA2             DC.L     0
RA3             DC.L     0
RA4             DC.L     0
RA5             DC.L     0
RA6             DC.L     0
```

図7.2　STARTシステム・コール

(4) WAKEUPシステム・コール

このシステム・コールは指定したタスクをウェイト（Wait）状態からレディ状態にするものです．これを呼ぶには，#20をD0レジスタに，パラメータ・ブロック・アドレスをA0レジスタにセットして，#1のトラップ命令を実行します(図7.3)．

```
                MOVE.L      #20, D0
                MOVE.L      #PRBLK, A0
                TRAP        #1
                  :
     PRBLK      EQU         *
                DC.L        'TSKZ'
                DC.L        0
```

図７．３　WAKEUPシステム・コール

(5)　**STDTIMシステム・コール**

このシステム・コールは，システムの日時の設定を行なうもので，設定する新しい日時（日付と時刻）のパラメータ・ブロック内に置きます．**STDTIM**システム・コールを呼ぶには，#73をＤ０レジスタに，パラメータ・ブロックのアドレスをＡ０レジスタにセットして，＃１のトラップ命令を実行すれば，パラメータ・ブロック内の内容がシステム日時として設定されます(**図7.4**)．

```
                MOVE.L      #73, D0
                LEA         PRBLK, A0
                TRAP        #1
                  :
     PRBLK      DC.L        85
                DC.L        $0
```

図７．４　STDTIMシステム・コール

以上，いくつかのシステム・コール例で見てきたように，レジスタにパラメータ，またはパラメータ・ブロックの先頭アドレスなどをセットして，

TRAP #1

命令で，トラップを発生させ，リアルタイム・マルチタスク・モニタにエントリします．

TRAP #〈トラップ・ベクタ番号〉………トラップを無条件に発生する

X	N	Z	V	C
–	–	–	–	–

コンディション・コードは，一切変化しません．

TRAP命令のマシン語フォーマットは，次のとおりです．

15	14	13	12	11	10	9	8	7	6	5	4	3 〜 0
0	1	0	0	1	1	1	0	0	1	0	0	トラップ・ベクタ番号

トラップ・ベクタ番号フィールドは，**TRAP**のベクタ番号を指定するフィールドで，4ビット長で0から15までの番号を指定することができます．

7.2 TRAPV命令とマシン語

TRAPV（TRAP if oVerflow set）命令は，オーバーフローフラグ（Vフラグ）がセット（"1"）のときトラップを発生する命令で，この命令を使ってオーバーフローが発生したかどうかのチェックを行なうことができます．**TRAPV**命令は，固有の例外ベクタ番号を持っており，その値は7です．

この例外ベクタ番号7を4倍して（7×4＝28）28番地が求まり，ここにオーバーフロー処理プログラムの先頭アドレスをセットしておいて，オーバーフローフラグのセットによって発生したトラップにより，オーバーフロー処理プログラムに制御を渡すことができます．

TRAPV……オペランドはなし．オーバーフローフラグ（Vフラグ）＝1でトラップが発生する．

X	N	Z	V	C
–	–	–	–	–

コンディション・コードは，TRAP命令と同様に，TRAPV命令で一切変化しません．

マシン語フォーマットは次に示すとおりです．

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

1ワードの4E76Hで，変化するフィールドは存在しません．

7.3　CHK命令とマシン語

CHK（CHecK register against bounds）命令は，境界チェックを行なう命令で，境界からはみ出る場合には，トラップを発生するようになっている命令です．デスティネーション・オペランドには，必ずデータ・レジスタDnを用い，これを"0"とソース・オペランドの実効アドレス<EA>の内容と比較し，大小関係により次のように動きます．

なお，オペレーション・サイズはワードです．

Dn＜0……………………ならばNフラグを"1"にセットして，トラップを発生する．

Dn＞(<EA>)…………ならばNフラグを"0"にリセットして，トラップを発生する．

0≦Dn≦(<EA>)……ならばNフラグは不定（どうなるかわからない）で，トラップは発生しない．

CHK　<EA>，Dn…Dn＜0でNフラグをセットして，トラップ発生．0≦Dn≦(<EA>)でNフラグは不定となり，トラップは発生せず次の命令を実行する．Dn＞(<EA>)でNフラグをリセットして，トラップを発生する．

グをリセットして，トラップを発生する．

X	N	Z	V	C
−	*	U	U	U

コンディション・コードは，まずNフラグがいろいろ影響を受け変化します．前述したように，

Dn＜**0**……………………ならばN＝1にセットされ，

Dn＞(<**EA**>)…………ならばN＝0にリセットされ，

0≦**Dn**≦(<**EA**>) ……ならばN＝不定

となります．

Z，V，Cフラグは不定で，どうなるかわからず，拡張フラグXは変化しません．

マシン語フォーマットは，次に示すとおりです．

15	14	13	12	11　　　9	8	7	6	5　　　0
0	1	0	0	レジスタ	1	1	0	実効アドレス

レジスタ・フィールドは，デスティネーション・オペランドで用いるデータ・レジスタの番号を指定するフィールドで，実効アドレス・フィールドはソース・オペランドのアドレッシング・モードを指定します．

第2部

68000の命令一覧

68000の

ニーモニック	オペランド 第１オペランド	第２オペランド	命令の機能	記述例
ABCD	データ・レジスタ (Dy) メモリ (−(Ay))	, データ・レジスタ (Dx) , メモリ (−(Ax))	(Add Binary Coded Decimal with extend) 第１オペランドの内容と，第２オペランドの内容と拡張ビットXの内容を加算し，結果を第２オペランドに格納する． 加算は**BCD**（２進化10進数）加算で行なわれる．	ABCD D0, D1 ABCD D2, D3 ABCD −(A0), −(A1) ABCD −(A2), −(A3)
ADD	<EA> Dn	, Dn , <EA>	(**ADD** binary) 第１オペランドの内容と第２オペランドの内容を加算し，結果を第２オペランドに格納する．	ADD.B BVAR, D0 ADD.W WVAR, D0 ADD.L LVAR, D1 ADD.B D1, D2 ADD D1, WVAR ADD.L D1, A1 ADD.W D2, (A2) ADD.L D3, (A1)+
ADDA	<EA>	, An	(**ADD** Address) 第１オペランドの内容と第２オペランドのアドレス・レジスタの内容を加算し，結果を第２オペランドのアドレス・レジスタに格納する．	ADDA D0, A1 ADDA.L LVAR, A3

命令一覧

（注）　1＝"1"にセットされる
0＝"0"にクリアされる
＊＝影響あり
U＝未定
－＝影響なし

<table>
<tr><th>オブジェクト・コード</th><th>オペレーション・サイズ</th><th>フラグ</th></tr>
<tr><td>
15 14 13 12 11 9 8 7 6 5 4 3 2 0
| 1 | 1 | 0 | 0 | レジスタ Rx | 1 | 0 | 0 | 0 | 0 | R/M | レジスタ Ry |

R/M＝0　……データ・レジスタ － データ・レジスタ
　＝1　……メモリ－メモリ
レジスタRx……第2オペランド（デスティネーション）のレジスタ番号
レジスタRy……第1オペランド（ソース）のレジスタ番号
</td><td>B</td><td>X N Z V C
＊ U ＊ U ＊</td></tr>
<tr><td>
15 14 13 12 11 9 8 6 5 0
| 1 | 1 | 0 | 1 | レジスタ | OPモード | 実効アドレス |

レジスタ……データ・レジスタ番号
OPモード……　B　W　L　オペレーション
000　001　010　(<Dn>)＋(<EA>)→<Dn>
100　101　110　(<EA>)＋(<Dn>)→<EA>
実効アドレス……<EA>のアドレッシング・モード
</td><td>B, W, L</td><td>X N Z V C
＊ ＊ ＊ ＊ ＊</td></tr>
<tr><td>
15 14 13 12 11 9 8 6 5 0
| 1 | 1 | 0 | 1 | レジスタ | OPモード | 実効アドレス |

レジスタ……第2オペランドのアドレス・レジスタ番号
OPモード……　W　L　オペレーション
011　111　(<An>)＋(<EA>)→<An>
（ワード・オペレーションのときは，第1オペランドのソースをロング・ワードに符号拡張して，32ビットで加算する）
実効アドレス……第1オペランド（ソース）のアドレッシング・モード
</td><td>W, L</td><td>X N Z V C
－ － － － －</td></tr>
</table>

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
ADDI	#〈即値〉,	〈EA〉	(**ADD** Immediate) 第1オペランドの即値データと第2オペランドの内容を加算し，結果を第2オペランドに格納する．	ADDI.B　#2, D1 ADDI.W　#10, D2 ADDI.L　#$3000, D0 ADDI　#2, WVAR1
ADDQ	#〈即値〉,	〈EA〉	(**ADD** Quick) 第1オペランドの即値データと第2オペランドの内容を加算し，結果を第2オペランドに格納する． ただし，即値データの範囲は1～8．	ADDQ　#1, D0 ADDQ.L　#8, LVAR
ADDX	データ・レジスタ(Dy) メモリ(−(Ay)),	データ・レジスタ, (Dx) メモリ(−(Ax))	(**ADD** eXtended) 第1オペランドの内容と，第2オペランドの内容と拡張ビットXの内容を加算し，結果を第2オペランドに格納する．	ADDX　D1, D0 ADDX.L　D3, D2 ADDX　−(A1), −(A0) ADDX.L　−(A5), −(A4)

<table>
<tr><th>オ ブ ジ ェ ク ト ・ コ ー ド</th><th>オペレーション・サイズ</th><th>フ ラ グ</th></tr>
<tr><td>
<table>
<tr><td>15</td><td></td><td></td><td></td><td></td><td></td><td></td><td>8</td><td>7 6</td><td>5 0</td><td>15 0</td><td>15 0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>サイズ</td><td>実効アドレス</td><td>ワード即値(16ビット)
バイト即値(8ビット)</td><td>ロング即値(前ワードも含めて32ビット)</td></tr>
</table>
サイズ……オペレーション・サイズ
B W L
00 01 10
実効アドレス……第2オペランドのアドレッシング・モード
</td><td>B，W，L</td><td>X N Z V C
* * * * *</td></tr>
<tr><td>
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11 9</td><td>8</td><td>7 6</td><td>5 0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>データ</td><td>0</td><td>サイズ</td><td>実効アドレス</td></tr>
</table>
データ……即値データが入り，0で8を，1～7で1～7を表わす．
サイズ……オペレーション・サイズ
B W L
00 01 10
実効アドレス……第2オペランドのアドレッシング・モード
</td><td>B，W，L</td><td>X N Z V C
* * * * *</td></tr>
<tr><td>
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11 9</td><td>8</td><td>7 6</td><td>5</td><td>4</td><td>3</td><td>2 0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>DESTレジスタ
(Rx)</td><td>1</td><td>サイズ</td><td>0</td><td>0</td><td>R/M</td><td>SRCレジスタ
(Ry)</td></tr>
</table>
DESTレジスタRx……第2オペランドのレジスタ番号
サイズ……………………オペレーション・サイズ
B W L
00 01 10
R/M＝0………………レジスターレジスタ
＝1………………メモリーメモリ
SRCレジスタRy ……第1オペランドのレジスタ番号
</td><td>B，W，L</td><td>X N Z V C
* * * * *</td></tr>
</table>

ニーモニック	オペランド 第1オペランド	 第2オペランド	命令の機能	記述例
AND	〈EA〉 Dn	, Dn , 〈EA〉	(**AND** logical) 第1オペランドと第2オペランドの論理積(**AND**)をとって、この結果を第2オペランド（デスティネーション）へ格納する。 キャリーフラグC、オーバーフローフラグVは、ゼロ・クリアされる。	AND　D0, D1 AND.B　BVAR, D0 AND.L　MASK, D0 AND.B　D1, (A1) AND.W　D0, (A1)+ AND.L　D2, LVAR
ANDI	#〈即値〉	, 〈EA〉	(**AND** Immediate) 第1オペランドの即値データと第2オペランドの内容の論理積(**AND**)をとって、この結果を第2オペランドへ格納する。 キャリーフラグC、オーバーフローフラグVは、ゼロ・クリアされる。 ステータス・レジスタへの**ANDI**は、バイト、ワードのオペレーション・サイズが可能で、バイトのときはステータス・レジスタSRの下位バイトのみが影響され、ワードのときはステータス・レジスタSRのすべてに対して実行される。	ANDI　#$00FF, D0 ANDI.L　#$0F0FFFFF, D1 ANDI.B　#0, SR ANDI　#$7FFF, SR ANDI　#4, WVAR
ASL	データ・レジスタ(Dx) #〈即値〉 メモリ(〈EA〉)	, データ・レジスタ(Dy) , データ・レジスタ(Dy)	(**Arithmetic Shift Left**) デスティネーション・オペランドの内容を、カウント・ビット分だけ左へ算術シフトする。 最上位ビットは、キャリーフラグCと拡張フラグXに入り、左へシフトして空になった下位のビットには、ゼロが入る。 シフトのときに符号が変化すると、オーバーフローフラグVがセットされる。 カウント・ビット数は、メモリ内容のシフトのときは常に1で、データ・レジスタ内容のシフトのときは、カウント・ビット数をデータ・レジスタまたは即値で指定し、おのおの次の範囲の値を指定することができる。 　データ・レジスタ……0～63 　即値……………………1～8 メモリ内容のシフトはワード・オペレーションのみ可。	ASL　D0, D1 ASL.B　D1, D2 ASL.L　D1, D0 ASL.B　#2, D0 ASL.L　#4, D1 ASL　BITPTN ASL　(A0)

<table>
<tr><th>オ　ブ　ジ　ェ　ク　ト　・　コ　ー　ド</th><th>オペレーション・サイズ</th><th>フ　ラ　グ</th></tr>
<tr><td>

15	14	13	12	11　9	8　6	5　0
1	1	0	0	レジスタ	OPモード	実効アドレス

レジスタ……データ・レジスタ番号
OPモード……　B　W　L　オペレーション
000　001　010　(<Dn>)∧(<EA>)→<Dn>
100　101　110　(<EA>)∧(<Dn>)→<EA>
実効アドレス……<EA> のアドレッシング・モード

</td><td>B, W, L</td><td>X N Z V C
− ＊ ＊ 0 0</td></tr>
<tr><td>

15	14	13	12	11	10	9	8	7　6	5　0	15　0	15　0
0	0	0	0	0	0	1	0	サイズ	実効アドレス	ワード即値(16ビット) バイト即値(8ビット)	ロング即値（前ワードも含めて32ビット)

サイズ……オペレーション・サイズ
B　W　L
00　01　10
実効アドレス……第2オペランドのアドレッシング・モード

</td><td>B, W, L</td><td>X N Z V C
− ＊ ＊ 0 0</td></tr>
<tr><td>

●レジスタ内容をシフトする場合：

15	14	13	12	11　9	8	7　6	5	4	3	2　0
1	1	1	0	カウント/ レジスタ	1	サイズ	i/r	0	0	レジスタ

カウント/レジスタ……i/r＝0のとき，シフトするカウント・ビット数が即値でセットされ，0で8を，1～7で1～7を表わす．
i/r＝1のとき，シフトするカウント・ビット数を保持するデータ・レジスタDxの番号がセットされる．
サイズ……………………オペレーション・サイズ
B　W　L
00　01　10
i/r…………………………i/r＝0のとき，シフトするカウント・ビット数が即値でセットされる．
i/r＝1のとき，シフトするカウント・ビット数がデータ・レジスタDxに保持される．
レジスタ ………………シフトするデータを保持するデータ・レジスタDyのレジスタ番号．

（続く）

</td><td>●レジスタ内容のシフト
B, W, L
●メモリ内容のシフト
Wのみ</td><td>X N Z V C
＊ ＊ ＊ ＊ ＊</td></tr>
</table>

ニーモニック	オペランド 第1オペランド	オペランド 第2オペランド	命令の機能	記述例
ASL (続き)			(図)	
ASR	データ・レジスタ (Dx) #<即値> メモリ(<EA>)	, データ・レジスタ (Dy) , データ・レジスタ (Dy)	(Arithmetic Shift Right) デスティネーション・オペランドの内容を，カウント・ビット分だけ右へ算術シフトする． 最下位ビットは，キャリーフラグCと拡張フラグXに入り，右へシフトして空になった上位のビットには，サイン・ビットがそのままシフトされて入り，サイン・ビットはそのままの値が残る． カウント・ビット数は，メモリ内容のシフトのときは常に1，データ・レジスタ内容のシフトのときは，カウント・ビット数をデータ・レジスタまたは即値で指定し，おのおの次の範囲の値を指定することができる． データ・レジスタ……0～63 即値……………………1～8 メモリ内容のシフトはワード・オペレーションのみ可． (図)	ASR D0, D1 ASR.B D1, D2 ASR.L D1, D0 ASR.B #2, D0 ASR #3, D0 ASR.L #4, D1 ASR BITPTN ASR (A1)

オブジェクト・コード	オペレーション・サイズ	フラグ

●メモリ内容をシフトする場合：

15	14	13	12	11	10	9	8	7	6	5〜0
1	1	1	0	0	0	0	1	1	1	実効アドレス

実効アドレス……メモリ〈EA〉のアドレッシング・モード

●レジスタ内容をシフトする場合：

15	14	13	12	11〜9	8	7〜6	5	4	3	2〜0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	0	0	レジスタ

カウント/レジスタ……i/r＝0のとき，シフトするカウント・ビット数が即値でセットされ，0で8を，1〜7で1〜7を表わす．

i/r＝1のとき，シフトするカウント・ビット数を保持するデータ・レジスタ番号がセットされる．

サイズ……オペレーション・サイズ

B	W	L
00	01	10

i/r……i/r＝0のとき，シフトするカウント・ビット数が即値でセットされる．

i/r＝1のとき，シフトするカウント・ビット数がデータ・レジスタDxに保持される．

レジスタ……シフトするデータを保持するデータ・レジスタ番号．

●メモリ内容をシフトする場合：

15	14	13	12	11	10	9	8	7	6	5〜0
1	1	1	0	0	0	0	0	1	1	実効アドレス

実効アドレス……メモリ〈EA〉のアドレッシング・モード

オペレーション・サイズ：

●レジスタ内容のシフト B，W，L

●メモリ内容のシフト Wのみ

フラグ：

X	N	Z	V	C
*	*	*	*	*

<table>
<tr><th rowspan="2">ニーモニック</th><th colspan="2">オペランド</th><th rowspan="2">命令の機能</th><th rowspan="2">記述例</th></tr>
<tr><th>第1オペランド</th><th>第2オペランド</th></tr>
<tr><td>Bcc</td><td>〈ラベル〉</td><td></td><td>(Branch Conditionally)
条件付ブランチ命令で，指定した条件（CC）が成立すれば（真ならば），オペランドで指定した〈ラベル〉へブランチし，成立しなければ（偽ならば），次の命令を実行する．
ブランチ条件とそのニーモニックは，次に示すとおりで，このニーモニックを Bcc の cc に代入して，条件付ブランチ命令のニーモニックが作られる．
<table>
<tr><th>ニーモニック</th><th>ブランチ条件</th></tr>
<tr><td>CC</td><td>キャリー・クリア
(carry clear)</td></tr>
<tr><td>CS</td><td>キャリー・セット
(carry set)</td></tr>
<tr><td>EQ</td><td>等しい (equal)</td></tr>
<tr><td>GE</td><td>大きい，または等しい
(greater or equal)</td></tr>
<tr><td>GT</td><td>大きい (greater)</td></tr>
<tr><td>HI</td><td>高い (high)</td></tr>
<tr><td>LE</td><td>小さい，または等しい
(less or equal)</td></tr>
<tr><td>LS</td><td>低い，または同じ
(low or same)</td></tr>
<tr><td>LT</td><td>小さい (less)</td></tr>
<tr><td>MI</td><td>負(マイナス) (minus)</td></tr>
<tr><td>NE</td><td>等しくない (not equal)</td></tr>
<tr><td>PL</td><td>正(プラス) (plus)</td></tr>
<tr><td>VC</td><td>オーバーフロー・クリア，
オーバーフローなし
(overflow clear, no overflow)</td></tr>
<tr><td>VS</td><td>オーバーフロー・セット，
オーバーフロー
(overflow set, overflow)</td></tr>
</table></td><td>BEQ　LABELA
BGE.S　LABELB
BVS　LABELOV</td></tr>
<tr><td>BCC</td><td>〈ラベル〉</td><td></td><td>(Branch if Carry Clear)
キャリーフラグ（C）＝0ならば，オペランドで指定したラベルへブランチし，（C）＝1ならば次の命令を実行する．</td><td>BCC　LABEL
BCC.S　LABELS</td></tr>
</table>

オブジェクト・コード	オペレーション・サイズ	フラグ

15	14	13	12	11 … 8	7 … 0	15 … 0
0	1	1	0	条件	DISP8	DISP16(このときDISP8＝0)

条件……ブランチ条件

ニーモニック	ブランチ条件	コード
CC	キャリー・クリア (carry clear)	0100
CS	キャリー・セット (carry set)	0101
EQ	等しい (equal)	0111
GE	大きい，または等しい (greater or equal)	1100
GT	大きい (greater)	1110
HI	高い (high)	0010
LE	小さい，または等しい (less or equal)	1111
LS	低い，または同じ (low or same)	0011
LT	小さい (less)	1101
MI	負(マイナス) (minus)	1011
NE	等しくない (not equal)	0110
PL	正(プラス) (plus)	1010
VC	オーバーフロー・クリア，オーバーフローなし (overflow clear, no overflow)	1000
VS	オーバーフロー・セット，オーバーフロー (overflow set, overflow)	1001

DISP8 ……8ビット・ディスプレースメント
DISP16……16ビット・ディスプレースメント

オペレーション・サイズ：B，W

フラグ：X N Z V C ／ － － － － －

15	14	13	12	11	10	9	8	7 … 0	15 … 0
0	1	1	0	0	1	0	0	DISP8	DISP16(このときDISP8＝0)

DISP8 ……8ビット・ディスプレースメント
DISP16……16ビット・ディスプレースメント

オペレーション・サイズ：B，W

フラグ：X N Z V C ／ － － － － －

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
BCS	〈ラベル〉		(Branch if Carry Set) キャリーフラグ(C)=1ならば，オペランドで指定したラベルへブランチし，(C)=0ならば次の命令を実行する．	BCS LABEL BCS.S LABELS
BEQ	〈ラベル〉		(Branch if EQual) ゼロ・フラグ(Z)=1ならば，オペランドで指定したラベルへブランチし，(Z)=0ならば次の命令を実行する．	BEQ LABEL BEQ.S LABELS
BGE	〈ラベル〉		(Branch if Greater or Equal) 大きい，または等しければ，すなわちネガティブ・フラグ(N)とオーバーフローフラグ(V)が等しければ((N)=(V))，オペランドで指定したラベルへブランチし，(N)≠(V)ならば次の命令を実行する．	BGE LABEL BGE.S LABELS
BGT	〈ラベル〉		(Branch if GreaTer) 大きければ，すなわちゼロ・フラグ(Z)=0で，かつネガティブ・フラグ(N)とオーバーフローフラグ(V)とが等しければ(すなわちN，Vともに"0"かまたは"1")，オペランドで指定したラベルへブランチし，(Z)=1または(N)≠(V)ならば，次の命令を実行する．	BGT LABEL BGT.S LABELS
BHI	〈ラベル〉		(Branch if HIgh) 高ければ(上にあれば)，すなわちキャリーフラグCとゼロ・フラグZがともに0ならば，オペランドで指定したラベルへブランチし，(C)=1または(Z)=1ならば，次の命令を実行する．	BHI LABEL BHI.S LABELS

オブジェクト・コード										オペレーション・サイズ	フラグ
15	14	13	12	11	10	9	8	7　0	15　0		
0	1	1	0	0	1	0	1	DISP8	DISP16（このときDISP8＝0）		
DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント										B, W	X N Z V C ― ― ― ― ―
15	14	13	12	11	10	9	8	7　0	15　0		
0	1	1	0	0	1	1	1	DISP8	DISP16（このときDISP8＝0）		
DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント										B, W	X N Z V C ― ― ― ― ―
15	14	13	12	11	10	9	8	7　0	15　0		
0	1	1	0	1	1	0	0	DISP8	DISP16（このときDISP8＝0）		
DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント										B, W	X N Z V C ― ― ― ― ―
15	14	13	12	11	10	9	8	7　0	15　0		
0	1	1	0	1	1	1	0	DISP8	DISP16（このときDISP8＝0）		
DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント										B, W	X N Z V C ― ― ― ― ―
15	14	13	12	11	10	9	8	7　0	15　0		
0	1	1	0	0	0	1	0	DISP8	DISP16（このときDISP8＝0）		
DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント										B, W	X N Z V C ― ― ― ― ―

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
BLE	〈ラベル〉		(Branch if Less or Equal) 小さいまたは等しければ，すなわち，ゼロ・フラグ（Z）＝1かまたはネガティブ・フラグNとオーバーフローフラグVが等しくなければ（すなわち（Z）＝1 or （N）≠（V）），オペランドで指定したラベルへブランチし，ゼロ・フラグ（Z）＝0で（N）＝（V）ならば（すなわち（Z）＝0 and （N）＝（V）），次の命令を実行する．	BLE　LABEL BLE.S　LABELS
BLS	〈ラベル〉		(Branch if Low or Same) 低い（下）または同じ（等しい）ならば，すなわちキャリーフラグ（C）＝1またはゼロ・フラグ（Z）＝1ならば，オペランドで指定したラベルへブランチし，（C），（Z）ともにゼロ（すなわち（C）＝0 and（Z）＝0）ならば，次の命令を実行する．	BLS　LABEL BLS.S　LABELS
BLT	〈ラベル〉		(Branch if Less) 小さいならば，すなわちネガティブ・フラグNとオーバーフローフラグVが等しくなければ（（N）≠（V）），オペランドで指定したラベルへブランチし，（N）＝（V）ならば次の命令を実行する．	BLT　LABEL BLT.S　LABELS
BMI	〈ラベル〉		(Branch if MInus) 負（マイナス）ならば，すなわちネガティブ・フラグ（N）＝1ならばオペランドで指定したラベルへブランチし，（N）＝0ならば次の命令を実行する．	BMI　LABEL BMI.S　LABELS
BNE	〈ラベル〉		(Branch if Not Equal) 等しくなければ，すなわちゼロ・フラグ（Z）＝0ならばオペランドで指定したラベルへブランチし，（Z）＝1ならば次の命令を実行する．	BNE　LABEL BNE.S　LABELS

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 10 9 8 7 0 15 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | DISP8 | DISP16（このときDISP8＝0）| DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント	B, W	X N Z V C – – – – –
15 14 13 12 11 10 9 8 7 0 15 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | DISP8 | DISP16（このときDISP8＝0）| DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント	B, W	X N Z V C – – – – –
15 14 13 12 11 10 9 8 7 0 15 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | DISP8 | DISP16（このときDISP8＝0）| DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント	B, W	X N Z V C – – – – –
15 14 13 12 11 10 9 8 7 0 15 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | DISP8 | DISP16（このときDISP8＝0）| DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント	B, W	X N Z V C – – – – –
15 14 13 12 11 10 9 8 7 0 15 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | DISP8 | DISP16（このときDISP8＝0）| DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント	B, W	X N Z V C – – – – –

ニーモニック	オペランド		命令の機能	記述例
	第 1 オペランド	第 2 オペランド		
BCLR	Dn #〈即値〉	,〈EA〉 ,〈EA〉	(test a **B**it and **CL**ea**R**) 第 2 オペランドの内容の，第 1 オペランドで指定したビットをテストし，ゼロかどうかでゼロ・フラグZが変化する．その後，テストしたビットをゼロ・クリアする． 第 2 オペランドがデータ・レジスタのとき，オペレーション・サイズはロング・ワードとなり，ビット番号は 0 ～31が用いられる．第 2 オペランドがメモリのとき，オペレーション・サイズはバイトとなり，ビット番号は 0 ～ 7 が用いられる． 第 1 オペランドでビット番号が指定され，それにはデータ・レジスタを用いる方法と，即値を用いる方法との 2 とおりが用意されている．	BCLR D0, D1 BCLR D0, FIELD BCLR #2, D2 BCLR #4, PORTA
BRA	〈ラベル〉		(**BR**anch **A**lways) オペランドで指定したラベルに無条件にブランチする． ディスプレースメントには，8 ビットと16ビットが用意されており，16ビットで表現可能(±32KB)な範囲内のラベルへブランチすることができる． ±128の範囲内のラベルへのブランチでは，8 ビットのディスプレースメントで表現可．	BRA LABEL BRA.S LABELS
BSET	Dn #〈即値〉	,〈EA〉 ,〈EA〉	(test a **B**it and **SET**) 第 2 オペランドの内容の，第 1 オペランドで指定したビットをテストし，ゼロかどうかでゼロ・フラグZが変化する．その後，テストしたビットを 1 にセットする． 第 2 オペランドがデータ・レジスタのとき，オペレーション・サイズはロング・ワードとなり，ビット番号は 0 ～31が用いられる． 第 2 オペランドがメモリのとき，オペレーション・サイズはバイトとなり，ビット番号は 0 ～ 7 が用いられる． 第 1 オペランドでビット番号が指定され，それにはデータ・レジスタを用いる方法と，即値を用いる方法との 2 とおりが用意されている．	BSET D0, D2 BSET D0, STAT BSET #1, D1 BSET #2, STATUS

オ ブ ジ ェ ク ト ・ コ ー ド	オペレーション・サイズ	フ ラ グ
●ビット番号をデータ・レジスタを用いて指定する場合： 15 14 13 12 11 9 8 7 6 5 0 | 0 | 0 | 0 | 0 | レジスタ | 1 | 1 | 0 | 実効アドレス | レジスタ…………ビット番号を格納するデータ・レジスタ番号 実効アドレス……〈EA〉のアドレッシング・モード	●データ・レジスタ内容のBCLR L ●メモリ内容のBCLR B	X N Z V C － － ＊ － －
●ビット番号を即値で直接指定する場合： 15 14 13 12 11 10 9 8 7 6 5 0 15 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 実効アドレス | ビ ッ ト 番 号 | 実効アドレス……〈EA〉のアドレッシング・モード ビット番号 ……ビット番号（即値）		
15 14 13 12 11 10 9 8 7 0 15 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | DISP8 | DISP16（このときDISP8＝0） | DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント	B，W	X N Z V C － － － － －
●ビット番号をデータ・レジスタを用いて指定する場合： 15 14 13 12 11 9 8 7 6 5 0 | 0 | 0 | 0 | 0 | レジスタ | 1 | 1 | 1 | 実効アドレス | レジスタ…………ビット番号を格納するデータ・レジスタ番号 実効アドレス……〈EA〉のアドレッシング・モード	●データ・レジスタ内容のBEST L ●メモリ内容のBEST B	X N Z V C － － ＊ － －
●ビット番号を即値で直接指定する場合： 15 14 13 12 11 10 9 8 7 6 5 0 15 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 実効アドレス | ビ ッ ト 番 号 | 実効アドレス……〈EA〉のアドレッシング・モード ビット番号 ……ビット番号（即値）		

ニーモニック	オペランド 第１オペランド	第２オペランド	命令の機能	記述例
BPL	〈ラベル〉		(**Branch if PLus**) ネガティブ・フラグ（N）＝０ならば，オペランドで指定したラベルへブランチし，（N）＝１ならば次の命令を実行する．	BPL　　LABEL BPL.S　LABELS
BVC	〈ラベル〉		(**Branch if oVerflow Clear**) オーバーフローフラグ（V）＝０ならば，オペランドで指定したラベルへブランチし，（V）＝１ならば次の命令を実行する．	BVC　　LABEL BVC.S　LABELS
BVS	〈ラベル〉		(**Branch if oVerflow Set**) オーバーフローフラグ（V）＝１ならば，オペランドで指定したラベルへブランチし，（V）＝０ならば次の命令を実行する．	BVS　　LABEL BVS.S　LABELS
BCHG	Dn #〈即値〉	，〈EA〉 ，〈EA〉	(test a **Bit** and **CHanGe**) 第２オペランドの内容の，第１オペランドで指定したビットをテストし，ゼロかどうかでゼロ・フラグZが変化する．その後，テストしたビット内容を反転する（すなわち"０"であれば"１"に，"１"であれば"０"にする）． 第２オペランドがデータ・レジスタのとき，オペレーション・サイズはロング・ワードとなり，ビット番号は０～31が用いられる． 第２オペランドがメモリのとき，オペレーション・サイズはバイトとなり，ビット番号は０～７が用いられる． 第１オペランドでビット番号が指定され，それにはデータ・レジスタを用いる方法と，即値を用いる方法との２とおりが用意されている。	BCHG　#7, STATUS BCHG　D0, STATUS BCHG　D0, D1 BCHG　#4, D2

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 10 9 8 7 0 15 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | DISP8 | DISP16(このときDISP8＝0) | DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント	B，W	X N Z V C － － － － －
15 14 13 12 11 10 9 8 7 0 15 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | DISP8 | DISP16(このときDISP8＝0) | DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント	B，W	X N Z V C － － － － －
15 14 13 12 11 10 9 8 7 0 15 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | DISP8 | DISP16(このときDISP8＝0) | DISP8 ……8ビット・ディスプレースメント DISP16……16ビット・ディスプレースメント	B，W	X N Z V C － － － － －
●ビット番号をデータ・レジスタを用いて指定する場合： 15 14 13 12 11 9 8 7 6 5 0 | 0 | 0 | 0 | 0 | レジスタ | 1 | 0 | 1 | 実効アドレス | レジスタ…………ビット番号を格納するデータ・レジスタ番号 実効アドレス…… 〈EA〉のアドレッシング・モード ●ビット番号を即値で直接指定する場合： 15 14 13 12 11 10 9 8 7 6 5 0 15 14 13 12 11 10 9 8 7 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 実効アドレス | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ビット番号 | 実効アドレス…… 〈EA〉のアドレッシング・モード ビット番号 ……ビット番号（即値）	●データ・レジスタ内容のBCHG L ●メモリ内容のBCHG B	X N Z V C － － ＊ － －

ニーモニック	オペランド		命令の機能	記述例
	第 1 オペランド	第 2 オペランド		
BSR	〈ラベル〉		(**B**ranch to **S**ub**R**outine) サブルーチンをコールする． **BSR** 命令の次の命令のアドレス(すなわちサブルーチンからの戻りアドレス)を，システム・スタックに退避し，オペランドで指定したラベル（サブルーチンの先頭ラベル）へブランチする．	BSR　SUBA BSR.S　PROCS
BTST	Dn #〈即値〉，	〈E A〉 〈E A〉	(**B**it **TeST**) 第 2 オペランドの内容の，第 1 オペランドで指定したビットをテストし，ゼロかどうかでゼロ・フラグ Z が変化する． 第 2 オペランドがデータ・レジスタのとき，オペレーション：サイズはロング・ワードとなり，ビット番号は 0 ～31が用いられる． 第 2 オペランドがメモリのとき，オペレーション・サイズはバイトとなり，ビット番号は 0 ～ 7 が用いられる． 第 1 オペランドでビット番号が指定され，それにはデータ・レジスタを用いる方法と，即値を用いる方法との 2 とおりが用意されている．	BTST　D0, D1 BTST　D0, FLAG BTST　#2, D1 BTST　#4, FLAG
CHK	〈E A〉，	Dn	(**CH**ec**K** register against bounds) 境界チェックを行ない，境界からはみ出すときはトラップを発生する． すなわち，第 2 オペランドのデータ・レジスタの下位16ビット（1 ワード）の内容と 0（下限値）とを比較し，さらに第 1 オペランドの内容（上限値）と比較する． データ・レジスタの下位16ビットの値が，0 より小かまたは第 1 オペランドの内容の上限値より大きいときは，トラップを発生し，例外処理を開始する． 0 ≦Dn≦(〈EA〉) のときは，トラップは発生せず，次の命令を実行する．	CHK　UPPER, D1

オブジェクト・コード	オペレーション・サイズ	フラグ

15	14	13	12	11	10	9	8	7 … 0	15 … 0
0	1	1	0	0	0	0	1	DISP8	DISP16（このときDISP8＝0）

DISP8 ……8ビット・ディスプレースメント
DISP16……16ビット・ディスプレースメント

オペレーション・サイズ	フラグ
B，W	X N Z V C － － － － －

●ビット番号をデータ・レジスタを用いて指定する場合：

15	14	13	12	11 … 9	8	7	6	5 … 0
0	0	0	0	レジスタ	1	0	0	実効アドレス

レジスタ…………ビット番号を格納するデータ・レジスタ番号
実効アドレス……〈EA〉のアドレッシング・モード

●ビット番号を即値で直接指定する場合：

15	14	13	12	11	10	9	8	7	6	5 … 0	15 … 0
0	0	0	0	1	0	0	0	0	0	実効アドレス	ビット番号

実効アドレス……〈EA〉のアドレッシング・モード
ビット番号　……ビット番号（即値）

オペレーション・サイズ	フラグ
●データ・レジスタ内容の**BTST** L ●メモリ内容の**BTST** B	X N Z V C － － ＊ － －

15	14	13	12	11 … 9	8	7	6	5 … 0
0	1	0	0	レジスタ	1	1	0	実効アドレス

レジスタ…………内容（下位16ビット）がチェックされるデータ・レジスタの番号
実効アドレス……上限値を保持する第1オペランド〈E A〉のアドレッシング・モード

オペレーション・サイズ	フラグ
W	X N Z V C － ＊ U U U

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
CLR	〈EA〉		(**CLeaR** an operand) 第1オペランドの内容をゼロ・クリアする。すなわち第1オペランドのビットをすべてゼロにクリアする。	CLR.B BVAR CLR.W WVAR CLR.W (A0) CLR.L D1
CMP	〈EA〉, Dn		(**CoMP**are) 2つのオペランドを比較する。 第2オペランドから第1オペランドを減算して、その結果に基づいてフラグを変更する。 オペランド内容はともに変わらない。	CMP.B (A0), D0 CMP.W MASK, D1 CMP.L (A2), D2 CMP.L D1, D2 CMP D0, D1
CMPA	〈EA〉, An		(**CoMP**are Address) 第2オペランドのアドレス・レジスタから第1オペランドを減算して、その結果に基づいてフラグを変更する。 オペランド内容はともに変わらない。	CMPA.W D0, A1 CMPA.L D1, A1 CMPA.L A2, A1 CMPA.L (A0), A2
CMPI	#〈即値〉, 〈EA〉		(**CoMP**are Immediate) 第2オペランドから第1オペランドの即値データを減算し、その結果に基づいてフラグを変更する。 オペランド内容は変わらない。	CMPI.B #1, D0 CMPI.W #8, D1 CMPI.W #$FFFE, D0 CMPI.L #$1F, D1 CMPI.L #2, LVAR

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 10 9 8 7 6 5 0 [0][1][0][0][0][0][1][0][サイズ][実効アドレス] サイズ…………オペレーション・サイズ B W L 00 01 10 実効アドレス……〈EA〉のアドレッシング・モード	B, W, L	X N Z V C − 0 1 0 0
15 14 13 12 11 9 8 6 5 0 [1][0][1][1][レジスタ][OPモード][実効アドレス] レジスタ………第2オペランドのデータ・レジスタ番号 OPモード …… B W L オペレーション 000 001 010 (〈Dn〉)−(〈EA〉) 実効アドレス……〈EA〉のアドレッシング・モード	B, W, L	X N Z V C − * * * *
15 14 13 12 11 9 8 6 5 0 [1][0][1][1][レジスタ][OPモード][実効アドレス] レジスタ………第2オペランドのアドレス・レジスタ番号 OPモード …… W L オペレーション 011 111 (〈An〉)−(〈EA〉) Wのとき，第1オペランドをワードからロング・ワードに符号拡張して，32ビットで演算を行なう． 実効アドレス……〈EA〉のアドレッシング・モード	W, L	X N Z V C − * * * *
15 14 13 12 11 10 9 8 7 6 5 0 15 0 15 0 [0][0][0][0][1][1][0][0][サイズ][実効アドレス][ワード即値(16ビット) バイト即値(8ビット)][ロング即値(前ワードも含めて32ビット)] サイズ…………オペレーション・サイズ B W L 00 01 10 実効アドレス……第2オペランド〈EA〉のアドレッシング・モード バイト即値，ワード即値，ロング即値……即値が入るフィールド	B, W, L	X N Z V C − * * * *

ニーモニック	オペランド		命令の機能	記述例
	第1 オペランド	第2 オペランド		
CMPM	メモリ ((Ay)+),	メモリ ((Ax)+)	(**CoMPare Memory**) 第2オペランドのメモリ内容から、第1オペランドのメモリ内容を減算して、その結果に基づいてフラグを変更する。 オペランドのメモリ内容はともに変わらない。 オペランドのメモリ・アドレッシング・モードは、ポスト・インクリメント・アドレス・レジスタ間接が用いられる。	CMPM.B (A0)+, (A1)+ CMPM.W (A2)+, (A3)+ CMPM.L (A4)+, (A5)+
DBcc	Dn ,	〈ラベル〉	(test condition, **D**ecrement and **B**ranch) まず最初に、ccで指定した条件が成立しているかどうかを調べ、成立していれば、次の命令を実行する。成立していなければ、データ・レジスタDnを−1して、その結果Dnが−1になったら(Dn=−1)、次の命令を実行し、−1でなければ(Dn≠−1)第2オペランドで指定したラベルへブランチする。 DBcc 命令 条件(CC)は成立しているか? YES NO (Dn)←(Dn)−1 (Dn)=−1 ? YES (Dn =−1) NO (Dn≠−1) ラベルへブランチ 次の命令を実行 条件ccのニーモニックは、次に示すとおりで、このニーモニックを**DBcc**のccに代入して、命令のニーモニックが作られる。 (続く)	DBEQ D0, LABEL DBGE D1, LABEL DBPL D2, LABEL DBVC D3, LABEL

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 9 8 7 6 5 4 3 2 0 | 1 | 0 | 1 | 1 | レジスタRx | 1 | サイズ | 0 | 0 | 1 | レジスタRy | レジスタRx(DEST.)……第2オペランドのアドレス・レジスタ番号 サイズ……オペレーション・サイズ B W L 00 01 10 レジスタRy(SRC) ……第1オペランドのアドレス・レジスタ番号	B, W, L	X N Z V C — * * * *
15 14 13 12 11 8 7 6 5 4 3 2 0 15 0 | 0 | 1 | 0 | 1 | 条件 | 1 | 1 | 0 | 0 | 1 | レジスタ | DISP16 | 条件……ccのコード	W	X N Z V C — — — — —

ニーモニック	ブランチ条件	コード
CC	キャリー・クリア (carry clear)	0100
CS	キャリー・セット (carry set)	0101
EQ	等しい (equal)	0111
F	常に偽，常に成立せず (never true)	0001
GE	大きい，または等しい (greater or equal)	1100
GT	大きい (greater)	1110
HI	高い (high)	0010
LE	小さい，または等しい (less or equal)	1111
LS	低い，または同じ (low or same)	0011
LT	小さい (less)	1101
MI	負(マイナス) (minus)	1011
NE	等しくない (not equal)	0110
PL	正(プラス) (plus)	1010
T	常に真，常に成立 (always true)	0000
VC	オーバーフロー・クリア，オーバーフローなし (overflow clear, no overflow)	1000
VS	オーバーフロー・セット，オーバーフロー (overflow set, overflow)	1001

ニーモニック	プランチ条件
CC	キャリー・クリア (carry clear)
CS	キャリー・セット (carry set)
EQ	等しい (equal)
F	常に偽, 常に成立せず (never true)
GE	大きい, または等しい (greater or equal)
GT	大きい (greater)
HI	高い (high)
LE	小さい, または等しい (less or equal)
LS	低い, または同じ (low or same)
LT	小さい (less)
MI	負(マイナス) (minus)
NE	等しくない (not equal)
PL	正(プラス) (plus)
T	常に真, 常に成立 (always true)
VC	オーバーフロー・クリア, オーバーフロなし (overflow clear, no overflow)
VS	オーバーフロー・セット, オーバーフロー (overflow set, overflow)

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
DBCC	Dn ,	〈ラベル〉	(test Carry Clear, Decrement and Branch) キャリーフラグ(C)=0ならば次の命令を実行する. (C)=1ならばデータ・レジスタDnを−1し, その結果Dnが−1になったら, 次の命令を実行し, −1でなければ指定したラベルヘブランチする.	DBCC D0, LABEL DBCC D5, LOOPA
DBCS	Dn ,	〈ラベル〉	(test Carry Set, Decrement and Branch) キャリーフラグ(C)=1ならば次の命令を実行する. (C)=0ならばデータ・レジスタDnを−1し, その結果Dnが−1になったら, 次の命令を実行し, −1でなければ指定したラベルヘブランチする.	DBCS D1, LABEL DBCS D6, LOOPA

オブジェクト・コード	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント		

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15′　0
0	1	0	1	0	1	0	0	1	1	0	0	1	レジスタ	DISP16

オブジェクト・コード	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C － － － － －

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	0	1	0	1	1	1	0	0	1	レジスタ	DISP16

オブジェクト・コード	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C － － － － －

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
DBEQ	Dn ,	〈ラベル〉	(test **EQ**ual, **D**ecrement and **B**ranch) ゼロ・フラグ(Z)=1ならば，次の命令を実行する． (Z)=0ならばデータ・レジスタDnを－1して，その結果Dnが－1になったら，次の命令を実行し，－1でなければ指定したラベルへブランチする．	DBEQ D0, LABEL DBEQ D5, LOOPA
DBF	Dn ,	〈ラベル〉	(test never true, **D**ecrement and **B**ranch) データ・レジスタDnを－1して，その結果Dnが－1になったら，次の命令を実行し，－1でなければ指定したラベルへブランチする．	DBF D1, LABEL DBF D6, LOOPA
DBGE	Dn ,	〈ラベル〉	(test **G**reater or **E**qual, **D**ecrement and **B**ranch) ネガティブ・フラグ(N)とオーバーフローフラグ(V)が等しければ((N)=(V))，次の命令を実行する． (N)≠(V)ならば，データ・レジスタDnを－1して，その結果Dnが－1になったら，次の命令を実行し，－1でなければ指定したラベルへブランチする．	DBGE D0, LABEL DBGE D5, LOOPA
DBGT	Dn ,	〈ラベル〉	(test **G**rea**T**er, **D**ecrement and **B**ranch) ゼロ・フラグ(Z)=0で，かつネガティブ・フラグ(N)とオーバーフローフラグ(V)とが等しければ(すなわちN，Vともに"0"かまたは"1")，次の命令を実行する． (Z)=1または(N)≠(V)ならば，データ・レジスタDnを－1して，その結果Dnが－1になったら，次の命令を実行し，－1でなければ指定したラベルへブランチする．	DBGT D0, LABEL DBGT D5, LOOPA
DBHI	Dn ,	〈ラベル〉	(test **HI**gh, **D**ecrement and **B**ranch) キャリーフラグCとゼロ・フラグZがともに0(ゼロ)ならば，次の命令を実行する． (C)=1または(Z)=1ならば，データ・レジスタDnを－1して，その結果Dnが－1になったら，次の命令を実行し，－1でなければ指定したラベルへブランチする．	DBHI D0, LABEL DBHI D5, LOOPA

オブジェクト・コード	オペレーション・サイズ	フラグ

15	14	13	12	11	10	9	8	7	6	5	4	3	2 … 0	15 … 0
0	1	0	1	0	1	1	1	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C - - - - -

15	14	13	12	11	10	9	8	7	6	5	4	3	2 … 0	15 … 0
0	1	0	1	0	0	0	1	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C - - - - -

15	14	13	12	11	10	9	8	7	6	5	4	3	2 … 0	15 … 0
0	1	0	1	1	1	0	0	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C - - - - -

15	14	13	12	11	10	9	8	7	6	5	4	3	2 … 0	15 … 0
0	1	0	1	1	1	1	0	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C - - - - -

15	14	13	12	11	10	9	8	7	6	5	4	3	2 … 0	15 … 0
0	1	0	1	0	0	1	0	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C - - - - -

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
DBLE	Dn ,	〈ラベル〉	(test Less or Equal, Decrement and Branch) ゼロ・フラグ(Z)=1かまたはネガティブ・フラグNとオーバーフローフラグVが等しくなければ(すなわち(Z)=1 or (N)≠(V)),次の命令を実行する. ゼロ・フラグ(Z)=0で(N)=(V)ならば(すなわち(Z)=0 and (N)=(V)), データ・レジスタDnを－1して,その結果Dnが－1になったら, 次の命令を実行し, －1でなければ指定したラベルへブランチする.	DBLE D0, LABEL DBLE D5, LOOPA
DBLS	Dn ,	〈ラベル〉	(test Low or Same, Decrement and Branch) キャリーフラグ(C)=1またはゼロ・フラグ(Z)=1ならば, 次の命令を実行する. (C), (Z)ともにゼロ(すなわち(C)=0 and (Z)=0)ならば, データ・レジスタDnを－1して,その結果Dnが－1になったら,次の命令を実行し,－1でなければ指定したラベルへブランチする.	DBLS D0, LABEL DBLS D5, LOOPA
DBLT	Dn ,	〈ラベル〉	(test Less, Decrement and Branch) ネガティブ・フラグNとオーバーフローフラグVが等しくなければ((N)≠(V)),次の命令を実行する. (N)=(V)ならばデータ・レジスタDnを－1して,その結果Dnが－1になったら,次の命令を実行し,－1でなければ指定したラベルへブランチする.	DBLT D0, LABEL DBLT D5, LOOPA
DBMI	Dn ,	〈ラベル〉	(test MInus, Decrement and Branch) ネガティブ・フラグ(N)=1ならば, 次の命令を実行する. (N)=0ならばデータ・レジスタDnを－1して,その結果Dnが－1になったら,次の命令を実行し,－1でなければ指定したラベルへブランチする.	DBMI D0, LABEL DBMI D5, LOOPA
DBNE	Dn ,	〈ラベル〉	(test Not Equal, Decrement and Branch) ゼロ・フラグ(Z)=0ならば,次の命令を実行する. (Z)=1ならばデータ・レジスタDnを－1して, その結果Dnが－1になったら,次の命令を実行し,－1でなければ指定したラベルへブランチする.	DBNE D0, LABEL DBNE D5, LOOPA

オブジェクト・コード	オペレーション・サイズ	フラグ

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	1	1	1	1	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C — — — — —

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	0	0	1	1	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C — — — — —

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	1	1	0	1	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C — — — — —

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	1	0	1	1	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C — — — — —

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	0	1	1	0	1	1	0	0	1	レジスタ	DISP16

説明	オペレーション・サイズ	フラグ
レジスタ……データ・レジスタ番号 DISP16 ……16ビット・ディスプレースメント	W	X N Z V C — — — — —

ニーモニック	オペランド 第１オペランド	第２オペランド	命令の機能	記述例
DBPL	Dn，〈ラベル〉		(test **PL**us, **D**ecrement and **B**ranch) ネガティブ・フラグ（N）＝0ならば，次の命令を実行する． (N)＝1ならばデータ・レジスタDnを－1して，その結果Dnが－1になったら，次の命令を実行し，－1でなければ指定したラベルへブランチする．	DBPL D0, LABEL DBPL D5, LOOPA
DBRA	Dn，〈ラベル〉		(**D**ecrement and **BRA**nch) データ・レジスタDnを－1して，その結果Dnが－1になったら，次の命令を実行し，－1でなければ指定したラベルへブランチする．	DBRA D0, LABEL DBRA D5, LOOPA
DBT	Dn，〈ラベル〉		(test always **T**rue, **D**ecrement and **B**ranch) 次の命令を実行する．	DBT D0, LABEL DBT D5, LOOPA
DBVC	Dn，〈ラベル〉		(test o**V**erflow **C**lear, **D**ecrement and **B**ranch) オーバーフローフラグ（V）＝0ならば，次の命令を実行する． (V)＝1ならば，データ・レジスタDnを－1して，その結果Dnが－1になったら，次の命令を実行し，－1でなければ指定したラベルへブランチする．	DBVC D0, LABEL DBVC D5, LOOPA
DBVS	Dn，〈ラベル〉		(test o**V**erflow **S**et, **D**ecrement and **B**ranch) オーバーフローフラグ（V）＝1ならば，次の命令を実行する． (V)＝0ならば，データ・レジスタDnを－1して，その結果Dnが－1になったら，次の命令を実行し，－1でなければ指定したラベルへブランチする．	DBVS D0, LABEL DBVS D5, LOOPA

オブジェクト・コード	オペレーション・サイズ	フラグ

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	1	0	1	0	1	1	0	0	1	レジスタ	DISP16

レジスタ……データ・レジスタ番号
DISP16 ……16ビット・ディスプレースメント

オペレーション・サイズ	X	N	Z	V	C
W	－	－	－	－	－

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	0	0	0	1	1	1	0	0	1	レジスタ	DISP16

レジスタ……データ・レジスタ番号
DISP16 ……16ビット・ディスプレースメント

オペレーション・サイズ	X	N	Z	V	C
W	－	－	－	－	－

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	0	0	0	0	1	1	0	0	1	レジスタ	DISP16

レジスタ……データ・レジスタ番号
DISP16 ……16ビット・ディスプレースメント

オペレーション・サイズ	X	N	Z	V	C
W	－	－	－	－	－

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	1	0	0	0	1	1	0	0	1	レジスタ	DISP16

レジスタ……データ・レジスタ番号
DISP16 ……16ビット・ディスプレースメント

オペレーション・サイズ	X	N	Z	V	C
W	－	－	－	－	－

15	14	13	12	11	10	9	8	7	6	5	4	3	2　0	15　0
0	1	0	1	1	0	0	1	1	1	0	0	1	レジスタ	DISP16

レジスタ……データ・レジスタ番号
DISP16 ……16ビット・ディスプレースメント

オペレーション・サイズ	X	N	Z	V	C
W	－	－	－	－	－

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
DIVS	<EA>，Dn		(**DIV**ide **S**igned) 符号付き除算，整数除算． 第2オペランドを第1オペランドで符号付き除算し，その結果を第2オペランドに格納する． 第1オペランドは16ビット長，第2オペランドは32ビット長で，32ビット／16ビットの符号付き除算の結果，商は第2オペランド32ビットの下位16ビットに，余りは上位16ビットに格納される． 第2オペランドDn（32ビット） 31　16 15　0 余り(上位16ビット) | 商(下位16ビット) 余りの符号は，被除数の符号と同じ． 本命令実行中，次のようなケースも生じる． ● 0(ゼロ)で割ると，トラップを発生する． ●命令完了前にオーバーフローが検出され，オーバーフローフラグVがセットされても，オペランド内容は変化しない．	DIVS NUM, D1 DIVS D0 , D1 DIVS (A0), D0 DIVS #6 , D0
DIVU	<EA>，Dn		(**DIV**ide **U**nsigned) 符号なし除算． 第2オペランドを第1オペランドで符号なし除算し，その結果を第2オペランドに格納する． 第1オペランドは16ビット長，第2オペランドは32ビット長で，32ビット／16ビットの符号なし除算の結果，商は第2オペランド32ビットの下位16ビットに，余りは上位16ビットに格納される． 第2オペランドDn(32ビット) 31　16 15　0 余り(上位16ビット) | 商(下位16ビット) ● 0(ゼロ)で割ると，トラップを発生する． ●命令完了前にオーバーフローが検出され，オーバーフローフラグVがセットされても，オペランド内容は変化しない．	DIVU CONST, D0 DIVU D0 , D1 DIVU (A0) , D0 DIVU #5 , D0

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 9 8 7 6 5 0 1 0 0 0 レジスタ 1 1 1 実効アドレス レジスタ…………第2オペランドのデータ・レジスタ番号 実効アドレス……〈EA〉のアドレッシング・モード	W	X N Z V C — * * * 0
15 14 13 12 11 9 8 7 6 5 0 1 0 0 0 レジスタ 0 1 1 実効アドレス レジスタ…………第2オペランドのデータ・レジスタ番号 実効アドレス……〈EA〉のアドレッシング・モード	W	X N Z V C — * * * 0

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
EOR	Dn	〈EA〉	(**Exclusive OR** logical) 第1オペランドと第2オペランドの排他的論理和をとり，その結果を第2オペランドへ格納する．	EOR.B D0, BITPTN EOR D1, (A0) EOR.L D1, D1
EORI	#〈即値〉	〈EA〉	(**Exclusive OR Immediate**) 第1オペランドの即値データと第2オペランドの排他的論理和をとり，その結果を第2オペランドへ格納する． ステータス・レジスタへの**EORI**オペレーションは，バイトのときはステータス・レジスタ**SR**の下位バイトのみが影響され，ワードのときは，ステータス・レジスタSRのすべてに対して実行され，特権命令となる．	EORI #$00FF, D0 EORI.L #$0F0FFFFF, D1 EORI.B #0, SR EORI #$7FFF, SR EORI #4, WVAR
EXG	Rx	Ry	(**EX**chan**G**e registers) 2つのレジスタ間で，そのレジスタ内容を交換する．レジスタにはデータ・レジスタ，アドレス・レジスタが指定でき，データ・レジスタ間での交換，アドレス・レジスタ間での交換，データ・レジスタとアドレス・レジスタ間での交換が可能で，32ビットで交換が行なわれる．	EXG D0, D1 EXG A0, A1 EXG D0, A1
EXT	Dn		(sign **EXT**end) データ・レジスタ中のバイト・データをワードに，ワードをロング・ワードに符号拡張する． オペレーション・サイズがWのときは，データ・レジスタ中のバイト・データの符号（ビット7）を，そのデータ・レジスタのビット（15～8）に転送し，バイトからワードに符号拡張する． オペレーション・サイズがLのときは，データ・レジスタ中のワード・データの符号（ビット15）を，そのデータ・レジスタのビット（31～16）に転送し，ワードからロング・ワードに符号拡張する．	EXT.W D0 EXT.L D1

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 9 8 6 5 0 | 1 | 0 | 1 | 1 | レジスタ | OPモード | 実効アドレス | レジスタ…………第1オペランドのデータ・レジスタ番号 OPモード …… B W L オペレーション 100 101 110 (〈EA〉)⊕(〈Dn〉)→〈EA〉 実効アドレス…… 〈EA〉のアドレッシング・モード	B, W, L	X N Z V C — * * 0 0
15 14 13 12 11 10 9 8 7 6 5 0 15 0 15 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | サイズ | 実効アドレス | ワード即値(16ビット) バイト即値(8ビット) | ロング即値(前ワードも含めて32ビット) | サイズ……………オペレーション・サイズ B W L 00 01 10 実効アドレス…… 〈EA〉のアドレッシング・モード バイト，ワード，ロング即値……即値フィールド	B, W, L	X N Z V C — * * 0 0
15 14 13 12 11 9 8 7 3 2 0 | 1 | 1 | 0 | 0 | レジスタRx | 1 | OPモード | レジスタRy | レジスタRx……第1オペランドのデータ・レジスタまたはアドレス・レジスタの番号．データ・レジスタとアドレス・レジスタ交換の場合，常にデータ・レジスタ番号． OPモード………データ・レジスタ間 アドレス・レジスタ間 データ・レジスタ・アドレス・レジスタ間 0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 レジスタRy……第2オペランドのデータ・レジスタまたはアドレス・レジスタの番号．データ・レジスタとアドレス・レジスタ交換の場合，常にアドレス・レジスタ番号	L	X N Z V C — — — — —
15 14 13 12 11 10 9 8 6 5 4 3 2 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | OPモード | 0 | 0 | 0 | レジスタ | OPモード……バイトからワードへの符号拡張 010 ワードからロング・ワードへの符号拡張 011 レジスタ………データ・レジスタ番号	W, L	X N Z V C — * * 0 0

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
JMP	〈EA〉		(**JuMP**) 指定したアドレスへジャンプする。	JMP START JMP (A0)
JSR	〈EA〉		(Jump to SubRoutine) 本命令の次の命令のアドレス（サブルーチンからの戻りアドレス）を，システム・スタックに退避，格納し，指定したアドレス（サブルーチンの先頭アドレス）へジャンプする。	JSR SUBA JSR (A0)
	〈EA〉	An	(Load Effective Address) 第1オペランドの実効アドレスを，第2オペランドのアドレス・レジスタにロードする。	LEA ABC1, A1 LEA (A0), A1
LINK	An,	#〈ディスプレースメント〉	(**LINK** and allocate) 第1オペランドで指定したアドレス・レジスタの現在の内容を，スタックにプッシュし，次にこの時点でのスタック・ポインタSPの値をアドレス・レジスタに転送し，最後にスタック・ポインタに第2オペランドで指定した〈ディスプレースメント〉の値を加算する。 **LINK** 命令を用いてスタックに領域を確保する。	LINK A0, #−8 LINK A1, #−32 LINK A5, #−64

オブジェクト・コード	オペレーションサイズ	フラグ

15	14	13	12	11	10	9	8	7	6	5～0
0	1	0	0	1	1	1	0	1	1	実効アドレス

実効アドレス……ジャンプ先アドレス指定

オペレーションサイズ	X	N	Z	V	C
——	—	—	—	—	—

15	14	13	12	11	10	9	8	7	6	5～0
0	1	0	0	1	1	1	0	1	0	実効アドレス

実効アドレス……ジャンプ先アドレス指定

オペレーションサイズ	X	N	Z	V	C
——	—	—	—	—	—

15	14	13	12	11～9	8	7	6	5～0
0	1	0	0	レジスタ	1	1	1	実効アドレス

レジスタ…………第2オペランドのアドレス・レジスタ番号
実効アドレス……〈EA〉のアドレッシング・モード

オペレーションサイズ	X	N	Z	V	C
L	—	—	—	—	—

15	14	13	12	11	10	9	8	7	6	5	4	3	2～0	15～0
0	1	0	0	1	1	1	0	0	1	0	1	0	アドレスレジスタ	DISP16

アドレス・レジスタ……第1オペランドのアドレス・レジスタ番号
DISP16 …………………16ビット・ディスプレースメント

オペレーションサイズ	X	N	Z	V	C
——	—	—	—	—	—

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
LSL	データ・レジスタ(Dx)， #〈即値〉， メモリ(〈EA〉)	データ・レジスタ(Dy) データ・レジスタ(Dy)	(**Logical Shift Left**) デスティネーション・オペランドの内容を，カウント・ビット分だけ左へ論理シフトする． 最上位ビットは，キャリーフラグCと拡張ビットXに入り，左へシフトして空になった下位のビットには，0(ゼロ)が入る． オーバーフローフラグVは0(ゼロ)にセットされる． カウント・ビット数は，メモリ内容のシフトのときは常に1で，データ・レジスタ内容のシフトのときは，カウント・ビット数をデータ・レジスタまたは即値で指定し，おのおの次の範囲の値を指定することができる． 　データ・レジスタ………0～63 　即　値　　　　………1～8 メモリ内容のシフトはワード・オペレーションのみ可． 31/15/7　0 C　X　データ・レジスタ　0 15　0 C　X　メモリ　0 (シフトできるのは1ビットのみ)	LSL　D0，D1 LSL.B　D1，D2 LSL.L　D1，D3 LSL.B　#3，D0 LSL　#4，D1 LSL.L　#5，D2 LSL　BITPTRN LSL　(A0)
LSR	データ・レジスタ(Dx)， #〈即値〉， メモリ(〈EA〉)	データ・レジスタ(Dy) ・レジスタ(Dy)	(**Logical Shift Right**) デスティネーション・オペランドの内容を，カウント・ビット分だけ右へ論理シフトする． 最下位ビットは，キャリーフラグCと拡張ビットXに入り，右へシフトして空になった上位のビットには，0(ゼロ)が入る． オーバーフローフラグVは，0(ゼロ)にセットされる． カウント・ビット数はメモリ内容のシフトのときは常に1で，データ・レジスタ内容のシフトのときはカウント・ビット数をデータ・レジスタまたは即値で指定し，おのおの次の範囲の値を指定することができる． 　データ・レジスタ………0～63 　即　値　　　　………1～8 メモリ内容のシフトはワード・オペレーションのみ可． (続く)	LSR　D0，D1 LSR.B　D1，D2 LSR.L　D1，D3 LSR.B　#3，D0 LSR　#4，D1 LSR.L　#5，D2 LSR　BITPTRN LSR　(A0)

<table>
<tr><th>オ ブ ジ ェ ク ト ・ コ ー ド</th><th>オペレーション・サイズ</th><th>フ ラ グ</th></tr>
<tr><td>

●レジスタ内容をシフトする場合：

15	14	13	12	11–9	8	7–6	5	4	3	2–0
1	1	1	0	カウント/レジスタ	1	サイズ	i/r	0	1	レジスタ

カウント/レジスタ……i/r＝0のとき，シフトするカウント・ビット数が即値でセットされ，0で8を，1～7で1～7を表わす．i/r＝1のとき，シフトするカウント・ビット数を保持するデータ・レジスタDxの番号がセットされる．
サイズ……………………オペレーション・サイズ
B　W　L
00　01　10
i/r…………………………i/r＝0のとき，シフトするカウント・ビット数が即値でセットされる．i/r＝1のとき，シフトするカウント・ビット数がデータ・レジスタDxに保持される．
レジスタ………………シフトするデータを保持するデータ・レジスタDyのレジスタ番号

●メモリ内容をシフトする場合：

15	14	13	12	11	10	9	8	7	6	5–0
1	1	1	0	0	0	1	1	1	1	実効アドレス

実効アドレス……メモリ〈EA〉のアドレッシング・モード

</td><td>●レジスタ内容のシフト
B，W，L
●メモリ内容のシフト
Wのみ</td><td>X N Z V C
＊ ＊ ＊ 0 ＊</td></tr>
<tr><td>

●レジスタ内容をシフトする場合：

15	14	13	12	11–9	8	7–6	5	4	3	2–0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	0	1	レジスタ

カウント/レジスタ……i/r＝0のとき，シフトするカウント・ビット数が即値でセットされ，0で8を，1～7で1～7を表わす．i/r＝1のとき，シフトするカウント・ビット数を保持するデータ・レジスタDxのレジスタ番号がセットされる．
サイズ……………………オペレーション・サイズ
B　W　L
00　01　10
i/r…………………………i/r＝0のとき，シフトするカウント・ビット数が即値でセットされる．i/r＝1のとき，シフトするカウント・ビット数がデータ・レジスタDxに保持される．
レジスタ………………シフトするデータを保持するデータ・レジスタDyのレジスタ番号

</td><td>●レジスタ内容のシフト
B，W，L
●メモリ内容のシフト
Wのみ</td><td>X N Z V C
＊ ＊ ＊ 0 ＊</td></tr>
</table>

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
			31/15/7 0 0 → データ・レジスタ → C, X 15 0 0 → メモリ → C, X (シフトできるのは1ビットのみ)	
MOVE	〈EA〉,	〈EA〉	(**MOVE** data from source to destination) 第1オペランドの内容を第2オペランドで指定するところへ転送する. 転送するときデータをチェックして，その結果によってコンディション・コードがセットされる. 変化するフラグはネガティブ・フラグNとゼロ・フラグZで，転送データが負のときN＝1にセットされ，転送データが0（ゼロ）のときZ＝1にセットされる. オーバーフローフラグVとキャリーフラグCは，ともにゼロ・クリアされる.	MOVE　DO, D1 MOVE.B　DO, D1 MOVE.L　DO, LVAR MOVE　WVAR, DO MOVE.L　AO, LVAR MOVE.L　D1, (A1) MOVE.B　DO, (A2)+ MOVE　(AO)+, (A1)+ MOVE.L　-(A1), -(A2) MOVE.L　AO, (A1) MOVE.L　DO, -4(AO) MOVE.W　(AO), DO MOVE.W　#1, D1 MOVE.W　A1, D1 MOVE.W　(A1), D1 MOVE.W　(A1)+, D1 MOVE.W　-(A1), D1 MOVE.W　$100(A1), D1 MOVE.W　$10(A1, DO), D1 MOVE.W　$100, D1 MOVE.W　$012000, D1 MOVE.W　#$1000, D1

オブジェクト・コード	オペレーションサイズ	フラグ
●メモリ内容をシフトする場合：		
(ビット配置 1)		
(ビット配置 2)	B, W, L	X N Z V C − * * 0 0

●メモリ内容をシフトする場合：

15	14	13	12	11	10	9	8	7	6	5　　　　0
1	1	1	0	0	0	1	0	1	1	実効アドレス

実効アドレス……メモリ〈EA〉のアドレッシング・モード

15	14	13　12	11　　9	8　　6	5　　3	2　　0
0	0	サイズ	デスティネーション (レジスタ)	デスティネーション (モード)	ソース (モード)	ソース (レジスタ)

サイズ……………………オペレーション・サイズ

B	W	L
01	11	10

デスティネーション……第2オペランドのアドレッシング・モード
ソース……………………第1オペランドのアドレッシング・モード

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
MOVE to **CCR**	〈EA〉	CCR	(**MOVE** to Condition Code Register) 第1オペランドの内容を，コンディション・コード・レジスタ**CCR**(ステータス・レジスタの下位8ビット)に転送する． 第1オペランドのサイズは，ワードで指定されるがこのワードの下位8ビットが**CCR**に転送される．	MOVE #0 ,CCR MOVE #4 ,CCR MOVE D0 ,CCR MOVE FLAG,CCR
MOVE to **SR**	〈EA〉	SR	(**MOVE** to Status Register) 第1オペランドの内容を，ステータス・レジスタ**SR**に転送する． コンディション・コードも含めて，ステータス・レジスタ**SR**のすべてのビットが影響される． 本命令は特権命令で，スーパバイザ状態でのみ実行可能．	MOVE #0 ,SR MOVE #$0A101,SR MOVE STAT ,SR

MOVE
from
SR
	SR	〈EA〉	(**MOVE** from Status Register)
			ステータス・レジスタ**SR**の内容を，第2オペランドで指定するところへ転送する．
				MOVE SR,D0
				MOVE SR,(A0)
				MOVE SR,(A1)+
				MOVE SR,SAVE

MOVE to/from **USP**	USP An	An USP	(**MOVE** to/from User Stack Pointer) ユーザースタック・ポインタ**USP**の内容をアドレス・レジスタへ，またはアドレス・レジスタの内容を**USP**へ転送する． オペレーション・サイズはロング・ワードで，本命令は特権命令である．	MOVE.L USP,A1 MOVE.L A0,USP

MOVEA	〈EA〉	An	(**MOVE** Address)
			第1オペランドの内容を，第2オペランドのアドレス・レジスタに転送する．
			オペレーション・サイズはワードとロング・ワードで，ワードのときは第1オペランドのワード・データをロング・ワードに符号拡張して，32ビットをアドレス・レジスタに転送する．
				MOVEA ADRS,A0
				MOVEA (A0),A2
				MOVEA.L LADRS,A1
				MOVEA.L (A2)+,A3

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 10 9 8 7 6 5 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 実効アドレス | 実効アドレス……第1オペランド〈EA〉のアドレッシング・モード	W	X N Z V C * * * * *
15 14 13 12 11 10 9 8 7 6 5 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 実効アドレス | 実効アドレス……第1オペランド〈EA〉のアドレッシング・モード	W	X N Z V C * * * * *
15 14 13 12 11 10 9 8 7 6 5 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 実効アドレス | 実効アドレス……第2オペランド〈EA〉のアドレッシング・モード	W	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 4 3 2 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | dr | レジスタ | dr ……転送方向 ＝0 ……アドレス・レジスタを**USP**へ転送 ＝1 ……**USP**をアドレス・レジスタへ転送 レジスタ……アドレス・レジスタ番号	L	X N Z V C — — — — —
15 14 13 12 11 9 8 6 5 3 2 0 | 0 | 0 | サイズ | デスティネーション (レジスタ) | 0 | 0 | 1 | ソース (モード) (レジスタ) | サイズ……オペレーション・サイズ W L 11 10 デスティネーション・レジスタ……第2オペランドのアドレス・レジスタ番号 ソース……第1オペランド〈EA〉のアドレッシング・モード	W, L	X N Z V C — — — — —

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		

MOVEM	〈レジスタ・リスト〉
	〈ＥＡ〉,
		,〈ＥＡ〉
		〈レジスタ・リスト〉
			(**MOVE M**ultiple registers)
			レジスタ・リストで指定した複数のレジスタの内容を，実効アドレス〈ＥＡ〉から始まるメモリに連続して転送したり，あるいはその逆に連続したメモリの内容を複数のレジスタへ転送する．
			転送で使用するレジスタの選択は，レジスタ・リストのマスク・フィールド中の対応するビットを1にセットすることにより行なわれる．
			オペレーション・サイズは，ワードまたはロング・ワードで，メモリのワード・データをレジスタに転送するときは，ワードはロング・ワードに符号拡張され，32ビットでレジスタに転送される．
			アドレッシング・モードは，制御アドレッシング・モード，プリ・デクリメント・アドレス・レジスタ間接，ポスト・インクリメント・アドレス・レジスタ間接が使用可能．
			制御アドレッシング・モードのとき，指定したメモリ・アドレスからアドレスの増える方向にあるメモリ内容と，指定した複数のレジスタとの間で転送を行なう．
			転送されるのは，レジスタ・リストで指定したレジスタで，転送する順序はデータ・レジスタD0→D7，アドレス・レジスタA0→A7の順である．
			プリ・デクリメント・アドレス・レジスタ間接のとき，複数のレジスタからメモリへの転送だけが可能で，指定したアドレスを前もって－2または－4だけデクリメントして，そこからアドレスの減る方向のメモリへ転送される．
			転送する順序は，アドレス・レジスタA7→A0，データ・レジスタD7→D0の順である（他と逆であることに注意）．
			ポスト・インクリメント・アドレス・レジスタ間接のとき，メモリから複数のレジスタへの転送だけが可能で，指定したアドレスからアドレスの増える方向のメモリ内容が転送される．
			転送する順序は，データ・レジスタD0→D7，アドレス・レジスタA0→A7の順である．
				MOVEM D0/A0-A1,$3000
				MOVEM D0-D7/A0-A7,-(SP)
				MOVEM (A0),D0/D2/A0
				MOVEM D0-D2/A0-A2,-(A6)
				MOVEM (A5)+,D0/D2/A0-A2

オブジェクト・コード	オペレーション・サイズ	フラグ

15	14	13	12	11	10	9	8	7	6	5 〜 0	15 〜 0
0	1	0	0	1	dr	0	0	1	Sz	実効アドレス	レジスタ・リスト・マスク

dr ………転送方向
 0……レジスタ→メモリ
 1……メモリ→レジスタ
Sz ………オペレーション・サイズ
 0……ワード・オペレーション
 1……ロング・ワード・オペレーション
実効アドレス……メモリ〈EA〉のアドレッシング・モード
レジスタ・リスト・マスク……転送で使用するレジスタを指定.
下位ビットのレジスタから上位ビットのレジスタへの順序で転送が行なわれる.
すなわち，下位ビットは最初に転送するレジスタに対応し，上位ビットは最後に転送するレジスタに対応する.
「命令の機能」でも述べたように，制御アドレッシング・モード，ポスト・インクリメント・アドレス・レジスタ間接アドレッシング・モードの場合と，プリ・デクリメント・アドレス・レジスタ間接アドレッシング・モードでは，転送順序が異なる.

制御アドレッシング・モードまたは
ポスト・インクリメント・アドレス・レジスタ間接アドレッシング・モードの場合：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

プリ・デクリメント・アドレス・レジスタ間接アドレッシング・モードの場合：

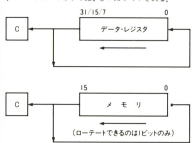
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7

オペレーション・サイズ	フラグ
W, L	X N Z V C – – – – –

ニーモニック	オペランド 第1オペランド	オペランド 第2オペランド	命令の機能	記述例

MOVEP	Dx	メモリ<d(Ay)>	(**MOVE** Peripheral data)
			データ・レジスタと1つおきのメモリ・アドレスとの間で、バイト・データを転送する。
			オペレーション・サイズは、ワードとロング・ワードが可能で、データ・レジスタ内容がメモリへ転送される順序は、データ・レジスタの最上位バイトが最初で、最下位バイトが最後となる。
			メモリの指定は、ディスプレースメント付アドレス・レジスタ間接のアドレッシング・モード(d(Ay))を用いる。
				MOVEP D0, 8(A0)
				MOVEP 4(A3), D5
				MOVEP.L D1, 10(A1)

	メモリ<d(Ay)>	Dx		
MOVEQ	#<即値>	Dn	(**MOVE** Quick) 第1オペランドの即値データを、第2オペランドのデータ・レジスタへ転送する。 オペレーション・サイズはロング・ワードのみで、オブジェクト・コード中にセットされた8ビットの即値データは、32ビットに符号拡張され、ロング・ワードとして第2オペランドのデータ・レジスタへ転送される。	MOVEQ #10, D0 MOVEQ #$10, D1
MULS	<EA>	Dn	(**MUL**tiply Signed) 符号付き乗算、整数乗算。 第1オペランドの内容(16ビット)と第2オペランドの内容(16ビット)を符号付き乗算し、結果を32ビットで第2オペランドのデータ・レジスタに格納する。 レジスタは下位16ビットを使用。	MULS D1, D0

<table>
<tr><th>オ ブ ジ ェ ク ト ・ コ ー ド</th><th>オペレーション・サイズ</th><th>フ ラ グ</th></tr>
<tr><td>
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11 9</td><td>8 6</td><td>5</td><td>4</td><td>3</td><td>2 0</td><td>15 0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>データ・レジスタ</td><td>OPモード</td><td>0</td><td>0</td><td>1</td><td>アドレス・レジスタ</td><td>DISP16</td></tr>
</table>
データ・レジスタ……データ・レジスタ番号
ＯＰモード…………オペレーション・サイズと転送方向
１００……メモリ→レジスタ，ワード転送
１０１……メモリ→レジスタ，ロング・ワード転送
１１０……レジスタ→メモリ，ワード転送
１１１……レジスタ→メモリ，ロング・ワード転送
アドレス・レジスタ……ディスプレースメント付アドレス・レジスタ間接のアドレッシング・モード（d(Ay)）で用いるアドレス・レジスタ番号．
DISP16 …………………16ビット・ディスプレースメント．
</td><td>W，L</td><td>X N Z V C
－ － － － －</td></tr>
<tr><td>
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11 9</td><td>8</td><td>7 0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>レジスタ</td><td>0</td><td>即値データ</td></tr>
</table>
レジスタ ……第２オペランドのデータ・レジスタ番号
即値データ……８ビットの即値データ（内部で32ビットに符号拡張される．）
</td><td>L</td><td>X N Z V C
－ * * 0 0</td></tr>
<tr><td>
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11 9</td><td>8</td><td>7</td><td>6</td><td>5 0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>レジスタ</td><td>1</td><td>1</td><td>1</td><td>実効アドレス</td></tr>
</table>
レジスタ ……第２オペランドのデータ・レジスタ番号
実効アドレス……〈EA〉のアドレッシング・モード
</td><td>W</td><td>X N Z V C
－ * * 0 0</td></tr>
</table>

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
MULU	<EA>	Dn	(**MUL**tiply **U**nsigned) 符号なし乗算. 第1オペランドの内容(16ビット)と,第2オペランドの内容(16ビット)を符号なし乗算し,結果を32ビットで第2オペランドのデータ・レジスタに格納する. レジスタは下位16ビットを使用.	MULU D1, D0
NBCD	<EA>		(**N**egate **B**inary **C**oded **D**ecimal with extend) 0から第1オペランドの内容と拡張ビットXの内容を減算(2進化10進数減算,BCD減算)し,結果を第1オペランドへ格納する. 0 −(<EA>)$_{10}$ −(X) → <EA>	NBCD D0 NBCD (A0) NBCD −(A1)
NEG	<EA>		(**NEG**ate, 2の補数の作成) 0から第1オペランドの内容を減算し,結果を第1オペランドへ格納する. 指定したオペランドの2の補数を作る. 0 −(<EA>) →<EA>	NEG.B D0 NEG.W D1 NEG (A0)+ NEG.L D2
NEGX	<EA>		(**NEG**ate with e**X**tend) 0から第1オペランドの内容と拡張ビットXの内容を減算し,結果を第1オペランドへ格納する. 0 −(<EA>)−(X) → <EA>	NEGX.B D0 NEGX D1 NEGX −(A0) NEGX.L D3

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 9 8 7 6 5 0 | 1 | 1 | 0 | 0 | レジスタ | 0 | 1 | 1 | 実効アドレス | レジスタ…………第2オペランドのデータ・レジスタ番号 実効アドレス……〈EA〉のアドレッシング・モード	W	X N Z V C — * * 0 0
15 14 13 12 11 10 9 8 7 6 5 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 実効アドレス | 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C * U * U *
15 14 13 12 11 10 9 8 7 6 5 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | サイズ | 実効アドレス | サイズ…………オペレーション・サイズ B W L 00 01 10 実効アドレス……〈EA〉のアドレッシング・モード	B, W, L	X N Z V C * * * * *
15 14 13 12 11 10 9 8 7 6 5 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | サイズ | 実効アドレス | サイズ…………オペレーション・サイズ B W L 00 01 10 実効アドレス……〈EA〉のアドレッシング・モード	B, W, L	X N Z V C * * * * *

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
NOP	なし		(No **O**P**eration**) なにもしないで次の命令に行く.	NOP
NOT	<EA>		(**NOT**, logical complement, 1の補数の作成) 指定したオペランドの1の補数を作る. すなわち, ビットをすべて反転させる.	NOT.B D0 NOT D1 NOT.L D0 NOT (A0) NOT WVAR
OR	<EA>, Dn Dn, <EA>		(inclusive **OR** logical) 第1オペランドと第2オペランドとの**OR**(論理和)をとって, 結果を第2オペランドへ格納する. キャリーフラグとオーバーフローフラグは, ともにリセットされる. (C)←0, (V)←0	OR.B D0, D1 OR D0, WVAR OR.L LVAR, D2
ORI	#<即値>, <EA>		(inclusive **OR** **I**mmediate) 第1オペランドの即値データと第2オペランドの論理和をとり, その結果を第2オペランドへ格納する. ステータス・レジスタへの**ORI**オペレーションは, バイトのときはステータス・レジスタ**SR**の下位バイトのみが影響され, ワードのときはステータス・レジスタ**SR**のすべてに対して実行され, 特権命令となる. キャリーフラグとオーバーフローフラグは, ともにリセットされる. (C)←0, (V)←0	ORI #$0FF, D0 ORI.L #$0F0FFFFF, D1 ORI.B #$0, SR ORI #$7FFF, SR ORI #8, WVAR

オ ブ ジ ェ ク ト ・ コ ー ド	オペレーション・サイズ	フ ラ グ
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |	——	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | サイズ | 実効アドレス | サイズ……………オペレーション・サイズ B W L 00 01 10 実効アドレス……〈EA〉のアドレッシング・モード	B, W, L	X N Z V C — ＊ ＊ 0 0
15 14 13 12 11 9 8 6 5 0 | 1 | 0 | 0 | 0 | レジスタ | OPモード | 実効アドレス | レジスタ…………データ・レジスタ番号 OPモード …… B W L オペレーション 000 001 010 (〈Dn〉)∨(〈EA〉)→〈Dn〉 100 101 110 (〈EA〉)∨(〈Dn〉)→〈EA〉 実効アドレス……〈EA〉のアドレッシング・モード	B, W, L	X N Z V C — ＊ ＊ 0 0
15 14 13 12 11 10 9 8 7 6 5 0 15 0 15 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | サイズ | 実効アドレス | ワード即値(16ビット) バイト即値(8ビット) | ロング即値(前ワードも含めて32ビット) | サイズ……………オペレーション・サイズ B W L 00 01 10 実効アドレス……〈EA〉のアドレッシング・モード バイト，ワード，ロング即値……即値フィールド	B, W, L	X N Z V C — ＊ ＊ 0 0

ニーモニック	オペランド 第1オペランド	オペランド 第2オペランド	命令の機能	記述例
PEA	〈EA〉		(**P**ush **E**ffective **A**ddress) オペランド〈EA〉の実効アドレスを計算し，これをスタックにロング・ワードでプッシュする．	PEA (A0) PEA 10(A1) PEA LVAR PEA $3000
RESET	なし		(**RESET** external devices) **RESET**命令を実行すると，リセット端子($\overline{RES}$)よりリセット信号(アクティブ"LOW")が124クロックの間出力される． これを用いて，外部デバイス，外部回路をリセットすることができる． プログラム・カウンタを除き，プロセッサの内部状態やレジスタは，一切影響を受けず，続いて次の命令から実行される．本命令は特権命令．	RESET
ROL	データ・レジスタ(Dx) #〈即値〉， メモリ(〈EA〉)	データ・レジスタ(Dy) データ・レジスタ(Dy)	(**RO**tate **L**eft without extend) デスティネーション・オペランドの内容を，カウント・ビット分だけ左へ回転(ローテート)する． 最上位から送り出されたビットは，キャリーフラグCと最下位ビットに入る． カウント・ビット数は，メモリ内容の回転のときは常に1で，データ・レジスタ内容の回転のときは，カウント・ビット数をデータ・レジスタまたは即値で指定し，おのおの次の範囲の値を指定することができる． 　データ・レジスタ………0～63 　即　値　　　　………1～8 メモリ内容の回転は，ワード・オペレーションのみ可． オーバーフローフラグVは，ゼロにクリアされる． [図] 31/15/7　0　C ← データ・レジスタ [図] 15　0　C ← メ モ リ　(ローテートできるのは1ビットのみ)	ROL D0, D1 ROL.B D1, D2 ROL.L D1, D3 ROL.B #3, D0 ROL #5, D2 ROL.L #7, D1 ROL STATUS ROL (A0) ROL 4(A1)

<table>
<tr><th>オブジェクト・コード</th><th>オペレーション・サイズ</th><th>フラグ</th></tr>
<tr><td>
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5 … 0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>実効アドレス</td></tr>
</table>
実効アドレス……〈EA〉のアドレッシング・モード
</td><td>L</td><td>X N Z V C
— — — — —</td></tr>
<tr><td>
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>
</td><td>——</td><td>X N Z V C
— — — — —</td></tr>
<tr><td>
●レジスタ内容を回転する場合：
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11 … 9</td><td>8</td><td>7 … 6</td><td>5</td><td>4</td><td>3</td><td>2 … 0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>カウント/レジスタ</td><td>1</td><td>サイズ</td><td>i/r</td><td>1</td><td>1</td><td>レジスタ</td></tr>
</table>
カウント/レジスタ……i/r＝0のとき，回転するカウント・ビット数が即値でセットされ，0で8を，1～7で1～7を表わす．
i/r＝1のとき，回転するカウント・ビット数を保持するデータ・レジスタDxの番号がセットされる．
サイズ……………………オペレーション・サイズ
B　W　L
00　01　10
i/r…………………………i/r＝0のとき，回転するカウント・ビット数が即値でセットされる．
i/r＝1のとき，回転するカウント・ビット数がデータ・レジスタDxに保持される．
レジスタ…………………回転(ローテート)するデータを保持するデータ・レジスタDyのレジスタ番号

●メモリ内容を回転する場合：
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5 … 0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>実効アドレス</td></tr>
</table>
実効アドレス……メモリ〈EA〉のアドレッシング・モード
</td><td>●レジスタ内容の回転
B，W，L
●メモリ内容の回転
Wのみ</td><td>X N Z V C
— ＊ ＊ 0 ＊</td></tr>
</table>

ニーモニック	オペランド 第1オペランド	 第2オペランド	命令の機能	記述例
ROR	データ・レジスタ(Dx) #〈即値〉, メモリ(〈EA〉)	, データ・レジスタ(Dy) データ・レジスタ(Dy)	(**RO**tate **R**ight without extend) デスティネーション・オペランドの内容を，カウント・ビット分だけ右へ回転(ローテート)する. 最下位から送り出されたビットは，キャリーフラグCと最上位ビットに入る. カウント・ビット数は，メモリ内容の回転のときは常に1で，データ・レジスタ内容の回転のときは，カウント・ビット数をデータ・レジスタまたは即値で指定し，おのおの次の範囲の値を指定することができる. データ・レジスタ………0～63 即　値　　　………1～8 メモリ内容の回転はワード・オペレーションのみ可. オーバーフローフラグVはゼロ・クリアされる. 31/15/7　0 データ・レジスタ　C 15　0 メ　モ　リ　C (ローテートできるのは1ビットのみ)	ROR　D0, D1 ROR.B　D1, D2 ROR.L　D1, D3 ROR.B　#3, D0 ROR　#5, D1 ROR.L　#7, D2 ROR　STATUS ROR　(A1) ROR　4(A2)
ROXL	データ・レジスタ(Dx) #〈即値〉, メモリ(〈EA〉)	, データ・レジスタ(Dy) データ・レジスタ(Dy)	(**RO**tate with e**X**tend **L**eft) デスティネーション・オペランドの内容を，カウント・ビット分だけ拡張フラグXも含めて，左へ回転(ローテート)する. 最上位から送り出されたビットは，キャリーフラグCと拡張フラグXに入り，Xは最下位ビットに入る. カウント・ビット数は，メモリ内容の回転のときは常に1で，データ・レジスタ内容の回転のときは，カウント・ビット数をデータ・レジスタまたは即値で指定し，おのおの次の範囲の値を指定することができる. データ・レジスタ………0～63 即　値　　　………1～8 メモリ内容の回転は，ワード・オペレーションのみ可. オーバーフローフラグVはゼロ・クリアされる. (続く)	ROXL　D0, D1 ROXL.B　D1, D2 ROXL.L　D1, D3 ROXL.B　#3, D0 ROXL　#5, D2 ROXL.L　#7, D1 ROXL　STATUS ROXL　(A0) ROXL　6(A1)

オブジェクト・コード	オペレーション・サイズ	フラグ

●レジスタ内容を回転する場合：

15	14	13	12	11–9	8	7–6	5	4	3	2–0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	1	1	レジスタ

カウント/レジスタ……i/r＝0のとき，回転するカウント・ビット数が即値でセットされ，0で8を，1～7で1～7を表わす．
i/r＝1のとき，回転するカウント・ビット数を保持するデータ・レジスタDxの番号がセットされる．

サイズ……オペレーション・サイズ

B	W	L
00	01	10

i/r……i/r＝0のとき，回転するカウント・ビット数が即値でセットされる．
i/r＝1のとき，回転するカウント・ビット数がデータ・レジスタDxに保持される．

レジスタ……回転（ローテート）するデータを保持するデータ・レジスタDyのレジスタ番号

●メモリ内容を回転する場合：

15	14	13	12	11	10	9	8	7	6	5–0
1	1	1	0	0	1	1	0	1	1	実効アドレス

実効アドレス……メモリ〈EA〉のアドレッシング・モード

オペレーション・サイズ：
●レジスタ内容の回転
B，W，L
●メモリ内容の回転
Wのみ

フラグ：

X	N	Z	V	C
−	＊	＊	0	＊

●レジスタ内容をXを含めて回転する場合：

15	14	13	12	11–9	8	7–6	5	4	3	2–0
1	1	1	0	カウント/レジスタ	1	サイズ	i/r	1	0	レジスタ

カウント/レジスタ……i/r＝0のとき，回転するカウント・ビット数が即値でセットされ，0で8を，1～7で1～7を表わす．
i/r＝1のとき，回転するカウント・ビット数を保持するデータ・レジスタDxの番号がセットされる．

サイズ……オペレーション・サイズ

B	W	L
00	01	10

i/r……i/r＝0のとき，回転するカウント・ビット数が即値でセットされる．
i/r＝1のとき，回転するカウント・ビット数がデータ・レジスタDxに保持される．

レジスタ……回転（ローテート）するデータを保持するデータ・レジスタDyのレジスタ番号

●メモリ内容をXを含めて回転する場合：

15	14	13	12	11	10	9	8	7	6	5–0
1	1	1	0	0	1	0	1	1	1	実効アドレス

実効アドレス……メモリ〈EA〉のアドレッシング・モード

オペレーション・サイズ：
●レジスタ内容の回転
B，W，L
●メモリ内容の回転
Wのみ

フラグ：

X	N	Z	V	C
−	＊	＊	0	＊

ニーモニック	オペランド 第1オペランド	オペランド 第2オペランド	命令の機能	記述例
ROXR	データ・レジスタ(Dx) #<即値>, メモリ(<EA>)	データ・レジスタ(Dy) データ・レジスタ(Dy)	(**RO**tate with e**X**tend **R**ight) デスティネーション・オペランドの内容を, カウント・ビット分だけ拡張フラグXも含めて, 右へ回転(ローテート)する. 最下位から送り出されたビットは, キャリーフラグCと拡張フラグXに入り, Xは最上位ビットに入る. カウント・ビット数は, メモリ内容の回転のときは常に1で, データ・レジスタ内容の回転のときは, カウント・ビット数をデータ・レジスタまたは即値で指定し, おのおの次の範囲の値を指定することができる. データ・レジスタ………0~63 即 値 ………1~8 メモリ内容の回転はワード・オペレーションのみ可. オーバーフローフラグVはゼロにクリアされる.	ROXR D0, D1 ROXR.B D1, D2 ROXR.L D1, D3 ROXR.B #3, D0 ROXR #5, D1 ROXR.L #7, D2 ROXR STATUS ROXR (A1) ROXR 4(A2)

オブジェクト・コード	オペレーション・サイズ	フラグ

●レジスタ内容をXを含めて回転する場合：

15	14	13	12	11–9	8	7–6	5	4	3	2–0
1	1	1	0	カウント/レジスタ	0	サイズ	i/r	1	0	レジスタ

カウント/レジスタ……i/r＝0のとき，回転するカウント・ビット数が即値でセットされ，0で8を，1～7で1～7を表わす．
i/r＝1のとき，回転するカウント・ビット数を保持するデータ・レジスタDxの番号がセットされる．

サイズ……オペレーション・サイズ

B	W	L
00	01	10

i/r……i/r＝0のとき，回転するカウント・ビット数が即値でセットされる．
i/r＝1のとき，回転するカウント・ビット数がデータ・レジスタDxに保持される．

レジスタ……回転（ローテート）するデータを保持するデータ・レジスタDyのレジスタ番号．

●メモリ内容をXを含めて回転する場合：

15	14	13	12	11	10	9	8	7	6	5–0
1	1	1	0	0	1	0	0	1	1	実効アドレス

実効アドレス……メモリ〈EA〉のアドレッシング・モード

オペレーション・サイズ：

●レジスタ内容の回転
B，W，L

●メモリ内容の回転
Wのみ

フラグ：

X	N	Z	V	C
—	＊	＊	0	＊

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
RTE	なし		(**Re**Turn from **E**xception) ステータス・レジスタSRとプログラム・カウンタPCの値を，システム・スタックからポップし，これをSR，PCにセットする． 新たなPC値から実行が再開される． 本命令は特権命令である．	RTE
RTR	なし		(**Re**Turn and **R**estore condition codes) コンディション・コードccとプログラム・カウンタPCの値を，スタックからポップし，これをコンディション・コード・レジスタCCR，PCにセットする． ステータス・レジスタSRの上位8ビットに影響はない． 新たなPC値から実行が再開される．	RTR
RTS	なし		(**Re**Turn from **S**ubroutine) プログラム・カウンタPCの値をスタックからポップし，これをPCにセットする．新たなPC値から実行が再開される．	RTS
SBCD	データ・レジスタ(Dy) メモリ(−(Ay))	データ・レジスタ(Dx) メモリ(−(Ax))	(**S**ubtract **B**inary **C**oded **D**ecimal with extend) 第2オペランドから，第1オペランドの内容と拡張ビットXの内容を減算（2進化10進数減算，BCD減算)し，結果を第2オペランドへ格納する．	SBCD D0, D1 SBCD D2, D3 SBCD −(A0), −(A1) SBCD −(A2), −(A3)

オブジェクト・コード	オペレーションサイズ	フラグ
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1	——	X N Z V C * * * * *
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1	——	X N Z V C * * * * *
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 0 1	——	X N Z V C — — — — —
15 14 13 12 | 11 10 9 | 8 7 6 5 4 | 3 | 2 1 0 1 0 0 0 | レジスタRx | 1 0 0 0 0 | R/M | レジスタRy レジスタRx……第2オペランドのレジスタ番号 R/M＝0のときデータ・レジスタ番号 R/M＝1のときアドレス・レジスタ番号 R/M…………レジスターレジスタかメモリーメモリかを指定 R/M＝0……レジスターレジスタ R/M＝1……メモリーメモリ レジスタRy……第1オペランドのレジスタ番号 R/M＝0のときデータ・レジスタ番号 R/M＝1のときアドレス・レジスタ番号	B	X N Z V C * U * U *

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
Scc	〈EA〉		(Set according to Condition Codes) ccで指定した条件が成立しているかどうかを調べ成立していれば，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする． 成立していなければ，そのバイトの全ビットを"0"にクリアする． 条件ccのニーモニックは，次に示すとおりで，このニーモニックを**Scc**のccに代入して，命令ニーモニックが作られる．	SEQ BVAR SGE (A0) SNE D2

ニーモニック	ブランチ条件
CC	キャリー・クリア (carry clear)
CS	キャリー・セット (carry set)
EQ	等しい (equal)
F	常に偽，常に成立せず (never true)
GE	大きい，または等しい (greater or equal)
GT	大きい (greater)
HI	高い (high)
LE	小さい，または等しい (less or equal)
LS	低い，または同じ (low or same)
LT	小さい (less)
MI	負(マイナス) (minus)
NE	等しくない (not equal)
PL	正(プラス) (plus)
T	常に真，常に成立 (always true)
VC	オーバーフロー・クリア， オーバーフローなし (overflow clear, no overflow)
VS	オーバーフロー・セット， オーバーフロー (overflow set, overflow)

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 8 7 6 5 0 0 | 1 | 0 | 1 | 条件 | 1 | 1 | 実効アドレス	B	X N Z V C — — — — —

条件……ccのコード

ニーモニック	ブランチ条件	コード
CC	キャリー・クリア (carry clear)	0100
CS	キャリー・セット (carry set)	0101
EQ	等しい (equal)	0111
F	常に偽，常に成立せず (never true)	0001
GE	大きい，または等しい (greater or equal)	1100
GT	大きい (greater)	1110
HI	高い (high)	0010
LE	小さい，または等しい (less or equal)	1111
LS	低い，または同じ (low or same)	0011
LT	小さい (less)	1101
MI	負(マイナス) (minus)	1011
NE	等しくない (not equal)	0110
PL	正(プラス) (plus)	1010
T	常に真，常に成立 (always true)	0000
VC	オーバーフロー・クリア，オーバーフローなし (overflow clear, no overflow)	1000
VS	オーバーフロー・セット，オーバーフロー (overflow set, overflow)	1001

実効アドレス……〈EA〉のアドレッシング・モード

ニーモニック	オペランド		命令の機能	記述例
	第 1 オペランド	第 2 オペランド		
SCC	〈EA〉		(Set according to Carry Clear) キャリーフラグ (C) =0ならば、オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする。 (C)=1ならば、そのバイトの全ビットを"0"にクリアする。	SCC BVAR SCC (A0)
SCS	〈EA〉		(Set according to Carry Set) キャリーフラグ (C) =1ならば、オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする。 (C)=0ならば、そのバイトの全ビットを"0"にクリアする。	SCS BVAR SCS (A0)
SEQ	〈EA〉		(Set according to EQual) ゼロ・フラグ (Z) =1ならば、オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする。 (Z)=0ならば、そのバイトの全ビットを"0"にクリアする。	SEQ BVAR SEQ (A0)
SF	〈EA〉		(Set according to never true) オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"0"にクリアする。	SF BVAR SF (A0)
SGE	〈EA〉		(Set according to Greater or Equal) ネガティブ・フラグ (N) とオーバーフローフラグ (V)が等しければ ((N)=(V))、オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする。 (N)≠(V)ならば、そのバイトの全ビットを"0"にクリアする。	SGE BVAR SGE (A0)

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 10 9 8 7 6 5〜0 0 1 0 1 0 1 0 0 1 1 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5〜0 0 1 0 1 0 1 0 1 1 1 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5〜0 0 1 0 1 0 1 1 1 1 1 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5〜0 0 1 0 1 0 0 0 1 1 1 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5〜0 0 1 0 1 1 1 0 0 1 1 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
SGT	〈EA〉		(Set according to GreaTer) ゼロ・フラグ(Z)=0で，かつネガティブ・フラグ(N)とオーバーフローフラグ(V)とが等しければ(すなわちN，Vともに"0"かまたは"1")，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする． (Z)=1または(N)≠(V)ならば，そのバイトの全ビットを"0"にクリアする．	SGT BVAR SGT (A0)
SHI	〈EA〉		(Set according to HIgh) キャリーフラグCとゼロ・フラグZがともに0(ゼロ)ならば，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする． (C)=1または(Z)=1ならば，そのバイトの全ビットを"0"にクリアする．	SHI BVAR SHI (A0)
SLE	〈EA〉		(Set accoding to Less or Equal) ゼロ・フラグ(Z)=1かまたはネガティブ・フラグNとオーバーフローフラグVが等しくなければ(すなわち(Z)=1 or (N)≠(V))，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする． ゼロ・フラグ(Z)=0で(N)=(V)ならば(すなわち(Z)=0 and (N)=(V))，そのバイトの全ビットを"0"にクリアする．	SLE BVAR SLE (A0)
SLS	〈EA〉		(Set according to Low or Same) キャリーフラグ(C)=1またはゼロ・フラグ(Z)=1ならば，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする． (C)，(Z)ともにゼロ(すなわち(C)=0 and (Z)=0)ならば，そのバイトの全ビットを"0"にクリアする．	SLS BVAR SLS (A0)
SLT	〈EA〉		(Set according to Less) ネガティブ・フラグNとオーバーフローフラグVが等しくなければ((N)≠(V))，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする． (N)=(V)ならば，そのバイトの全ビットを"0"にクリアする．	SLT BVAR SLT (A0)

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 10 9 8 7 6 5 0 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 0 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 0 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 0 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 0 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
SMI	〈EA〉		(**S**et according to **MI**nus) ネガティブ・フラグ(N)＝1ならば，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを‘1’にセットする． (N)＝0ならば，そのバイトの全ビットを‘0’にクリアする．	SMI BVAR SMI (AO)
SNE	〈EA〉		(**S**et according to **N**ot **E**qual) ゼロ・フラグ(Z)＝0ならば，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを‘1’にセットする． (Z)＝1ならば，そのバイトの全ビットを‘0’にクリアする．	SNE BVAR SNE (AO)
SPL	〈EA〉		(**S**et according to **PL**us) ネガティブ・フラグ(N)＝0ならば，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを‘1’にセットする． (N)＝1ならば，そのバイトの全ビットを‘0’にクリアする．	SPL BVAR SPL (AO)
ST	〈EA〉		(**S**et according to always **T**rue) オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを‘1’にセットする．	ST BVAR ST (AO)
SVC	〈EA〉		(**S**et according to o**V**erflow **C**lear) オーバーフローフラグ(V)＝0ならば，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを‘1’にセットする． (V)＝1ならば，そのバイトの全ビットを‘0’にクリアする．	SVC BVAR SVC (AO)

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 10 9 8 7 6 5 … 0 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 … 0 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 … 0 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 … 0 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 … 0 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —

ニーモニック	オペランド 第1オペランド	第2オペランド	命令の機能	記述例
SVS	〈EA〉		(**S**et according to o**V**erflow **S**et) オーバーフローフラグ（V）＝1ならば，オペランドの実効アドレス〈EA〉によって指定されるバイトの全ビットを"1"にセットする． （V）＝0ならば，そのバイトの全ビットを"0"にクリアする．	SVS BVAR SVS (A0)
STOP	＃〈即値データ〉		(load status register and **STOP**) オペランドで指定した即値データをワード（16ビット）でステータス・レジスタSRへ転送したのち，CPUを停止する． CPUは，命令のフェッチ，実行を停止するが，プログラム・カウンタPCは更新され，次の命令をポイントする． 何もなければ，CPUはストップしたままだが，トレース，割込み，リセット例外処理要因が生じると，停止を抜け出し，実行を再開する． STOP命令が実行を開始したとき，Tビットが1なら，トレース例外処理が開始される． 即値データでセットされた優先度より高い優先度の割込み要求が入力すると，割込み例外処理になるが，それ以外の割込み要求は受けつけられない． 外部リセットで，常にリセット例外処理が起動される． 本命令は特権命令である．	STOP #$7F00
SUB	〈EA〉，Dn		(**SUB**tract binary) 第2オペランドから第1オペランドを減算し，結果を第2オペランドへ格納する．	SUB.B BVAR, D0 SUB.W WVAR, D0 SUB.L LVAR, D1
	Dn，〈EA〉			SUB.B D1, D2 SUB D1, WVAR SUB.L D1, A1 SUB.W D1, (A2) SUB.L D3, (A1)+

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 10 9 8 7 6 5 0 0 1 0 1 1 0 0 1 1 1 | 実効アドレス 実効アドレス……〈EA〉のアドレッシング・モード	B	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 15 0 0 1 0 0 1 1 1 0 0 1 1 1 0 0 1 0 | 即値データ 即値データ……ステータス・レジスタSRに転送する即値データ・フィールド.	——	X N Z V C * * * * *
15 14 13 12 11 9 8 6 5 0 1 1 0 1 | レジスタ | OPモード | 実効アドレス レジスタ…………データ・レジスタ番号 OPモード……… B W L オペレーション 000 001 010 (〈Dn〉)−(〈EA〉)→〈Dn〉 100 101 110 (〈EA〉)−(〈Dn〉)→〈EA〉 実効アドレス……〈EA〉のアドレッシング・モード	B, W, L	X N Z V C * * * * *

ニーモニック	オペランド 第１オペランド	第２オペランド	命令の機能	記述例
SUBA	〈EA〉	An	(**SUB**tract Address) 第２オペランドのアドレス・レジスタから第１オペランドを減算し，結果を第２オペランドのアドレス・レジスタへ格納する．	SUBA D0, A1 SUBA.L LVAR, A3
SUBI	#〈即値〉	〈EA〉	(**SUB**tract Immediate) 第２オペランドから第１オペランドの即値データを減算し，結果を第２オペランドへ格納する．	SUBI.B #2, D1 SUBI.W #10, D2 SUBI.L #$3000, D0 SUBI #3, WVAR
SUBQ	#〈即値〉	〈EA〉	(**SUB**tract Quick) 第２オペランドから第１オペランドの即値データを減算し，結果を第２オペランドへ格納する． ただし，即値データの範囲は１～８．	SUBQ #1, D0 SUBQ.L #8, LVAR

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 9 8 6 5 0 | 1 | 0 | 0 | 1 | レジスタ | OPモード | 実効アドレス | レジスタ…………第2オペランドのアドレス・レジスタ番号 OPモード……… W　L　オペレーション 011　111　(<An>)−(<EA>)→<An> (ワード・オペレーションのときは，第1オペランドのソースをロング・ワードに符号拡張して，32ビットで減算する) 実効アドレス……第1オペランド〈EA〉のアドレッシング・モード	W，L	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 0 15 0 15 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | サイズ | 実効アドレス | ワード即値(16ビット) バイト即値(8ビット) | ロング即値(前ワードも含めて32ビット) | サイズ…………オペレーション・サイズ B　W　L 00　01　10 実効アドレス……第2オペランドのアドレッシング・モード(アドレス・レジスタ直接は不可) バイト，ワード，ロング即値……即値フィールド	B，W，L	X N Z V C * * * * *
15 14 13 12 11 9 8 7 6 5 0 | 0 | 1 | 0 | 1 | データ | 1 | サイズ | 実効アドレス | データ…………即値データが入り，0で8を，1〜7で1〜7を表わす． サイズ…………オペレーション・サイズ B　W　L 00　01　10 実効アドレス……第2オペランドのアドレッシング・モード	B，W，L	X N Z V C * * * * *

ニーモニック	オペランド 第１オペランド	第２オペランド	命令の機能	記述例

SUBX	データ・レジスタ (Dy)
	メモリ (−(Ay))
		, データ・レジスタ (Dx)
		, メモリ (−(Ax))
			(**SU**Btract with e**X**tend)
			第２オペランドから、第１オペランドと拡張ビットXを減算し、結果を第２オペランドへ格納する。
			（第２オペランド）−（第１オペランド）−（Ｘ）→第２オペランド
				SUBX.B D0, D1
				SUBX D2, D3
				SUBX.L D1, D2
				SUBX −(A1), −(A2)
				SUBX.B −(A0), −(A1)
				SUBX.L −(A4), −(A5)

SWAP	Dn		(**SWAP** register halves) オペランドで指定したデータ・レジスタの上位ワードと下位ワードの内容を入れ換える。	SWAP D0 SWAP D5
TAS	〈ＥＡ〉		(**T**est **A**nd **S**et an operand) 第１オペランドで指定したバイト・オペランドをテストして、その結果により、ネガティブ・フラグNとゼロ・フラグZをセットする。そして、バイト・オペランドの最上位ビットを“１”にセットする。 **TAS**命令は、リード・モディファイ・ライト・サイクルを使用しており、リードとライトは、連続して実行される。 本命令はマルチ・プロセッサ用の命令である。	TAS SEMA
TRAP	#〈トラップ・ベクタ番号〉		(**TRAP**) 無条件にトラップを発生させ、プロセッサは例外処理を開始する。 トラップ・ベクタ番号は０〜15を用いることができ、これらは例外ベクタ番号32〜47に対応する。 （例外ベクタ番号×４）番地がポイントするところから、トラップ処理ルーチンが始まる。	TRAP #1 TRAP #13

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 9 8 7 6 5 4 3 2 0 1 | 0 | 0 | 1 | DESTレジスタ (Rx) | 1 | サイズ | 0 | 0 | R/M | SRCレジスタ (Ry) DESTレジスタRx……第2オペランドのレジスタ番号 サイズ……………………オペレーション・サイズ B W L 00 01 10 R/M＝0………………レジスターレジスタ ＝1………………メモリーメモリ SRCレジスタRy ……第1オペランドのレジスタ番号	B, W, L	X N Z V C * * * * *
15 14 13 12 11 10 9 8 7 6 5 4 3 2 0 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | レジスタ レジスタ……データレジスタ番号	W	X N Z V C — * * 0 0
15 14 13 12 11 10 9 8 7 6 5 0 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 実効アドレス 実効アドレス……バイト・オペランド〈EA〉のアドレッシング・モード	B	X N Z V C — * * 0 0
15 14 13 12 11 10 9 8 7 6 5 4 3 0 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | トラップ・ベクタ番号 トラップ・ベクタ番号……0から15までのトラップ・ベクタ番号を指定	——	X N Z V C — — — — —

ニーモニック	オペランド		命令の機能	記述例
	第1オペランド	第2オペランド		
TRAPV	なし		(**TRAP** on o**V**erflow) コンディション・コードのオーバーフローフラグVが"1"のとき，トラップを発生し，プロセッサは例外処理を開始する． **TRAPV**命令はオペランドがなく，例外ベクタ番号は7であり，28番地がポイントするアドレスからオーバーフロー処理ルーチンが開始する． オーバーフローフラグVが"0"のときは，次の命令を実行する．	TRAPV
TST	〈EA〉		(**TeST** an operand) オペランド内容を0（ゼロ）と比較し，その結果によってコンディション・コードがセットされる． オペランド内容は変化しない．	TST D0 TST.B D1 TST.L D1 TST (A0) TST.L LVAR TST.W COUNT
UNLK	An		(**UNLinK**) オペランドで指定したアドレス・レジスタの内容をスタック・ポインタSPに転送し，次にスタックからポップしたロング・ワードをアドレス・レジスタに転送する． **LINK**命令と対で用いられ，**LINK**命令でスタック上に確保した領域を解放する．	UNLK A0 UNLK A1

オブジェクト・コード	オペレーション・サイズ	フラグ
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 0	——	X N Z V C — — — — —
15 14 13 12 11 10 9 8 7 6 5 0 0 1 0 0 1 0 1 0 | サイズ | 実効アドレス サイズ……………オペレーション・サイズ B W L 00 01 10 実効アドレス……オペランド〈EA〉のアドレッシング・モード	B, W, L	X N Z V C — ＊ ＊ 0 0
15 14 13 12 11 10 9 8 7 6 5 4 3 2 0 0 1 0 0 1 1 1 0 0 1 0 1 1 | アドレスレジスタ アドレス・レジスタ……アドレス・レジスタ番号	——	X N Z V C — — — — —

参 考 文 献

(1) MC68000 16ビット・マイクロコンピュータ・ユーザーズマニュアル
(日本モトローラ社)
(2) MC68020 32ビット・マイクロプロセッサ・ユーザーズマニュアル
(モトローラ社)
(3) 68000アセンブリ言語マニュアル (日立製作所)
(4) 喜田他 著「68000マイクロコンピュータ」(丸善)
(5) RMS68Kユーザース・ガイド(日本モトローラ社)

索　引

五十音順

〈あ　行〉
アドレス・レジスタ……10
アドレス・レジスタ間接……15
アドレス・レジスタ直接……15
アドレッシング・モード……14
アブソリュート・ショート……17
アブソリュート・ロング……17
インデックス・ディスプレースメント付
　アドレス・レジスタ間接……17
インデックス・ディスプレースメント付
　プログラム・カウンタ相対……18
インデックス・レジスタ番号……26
インプライド……19
1の補数の作成……242
エクステンド・フラグ……13
オーバーフローフラグ……13
オペランド……210
オペレーション・ワード……20
オペレーティング・システム・コール……183
〈か　行〉
カウント/レジスタ・フィールド……104
回転命令……114
回転命令のマシン語……122
加減算命令のマシン語……46
加算命令……40
キャリーフラグ……13
境界チェック……183
共有メモリ……88
クイック即値……19
クリア命令……73
クリア命令のマシン語……73
減算命令……45
交換……226
〈さ　行〉
サイズ・フィールド……23
サイン・ビット……198
サブルーチンの呼び出し命令のマシン語……166
最下位ビット(LSB)……105
最上位ビット(MSB)……59
再入可能(リエントラント)……173
算術演算命令……31
システム・コール……184
システム・スタック……210
システム・バイト……13
システム・フラグ……13
シフト命令……102
シフト命令のマシン語……110
シングル・ステップ……14
自己再帰(セルフ・リカーシブ)……173
実行……260
実効アドレス……21
実効アドレス・フィールド……104
条件付分岐命令……146
乗算命令……56
乗除算命令のマシン語……60
除算命令……58
スタック・ポインタ……10
ステータス・レジスタ……10
スーパバイザ・コール……183
スーパバイザ状態フラグ……14
スーパバイザ・スタック・ポインタ(SSP)……12
セマフォーオペレーション……88
ゼロ・フラグ……13
整数乗算……238
ソース・オペランド……23
ソース実効アドレス・オペランド……20
即値……18
即値オペランド……20
〈た　行〉
第1オペランド……192
第2オペランド……192
チップ集積度……3
テスト・アンド・セット命令……88
テスト・アンド・セット命令のマシン語……91
テスト命令……73
テスト命令のマシン語……74
デスティネーション・オペランド……23
デスティネーション実効アドレス・
　オペランド……20
ディスプレースメント……26, 55
ディスプレースメント付
　アドレス・レジスタ間接……16
ディスプレースメント付
　プログラム・カウンタ相対……18
データ転送命令……31, 35
データ転送命令のマシン語……37
データ・レジスタ……10
データ・レジスタ直接……14
データ・レジスタ内容……244
停止……260
トレース……14
トレース・モード・フラグ……14
トラップ……58, 183
トラップ処理ルーチン……184
トラップ発生命令……183
トラップ・ベクタ番号……184
特権命令……226
〈な　行〉
2進化10進数……192
2進化10進数減算……240
2の補数の作成……240
ネガティブ・フラグ……13
〈は　行〉
バイト……35
バイト・オペレーション……23
バイト即値……42
バイト・レジスタ……10
排他的論理和命令……79

ビット操作命令……………………………31,132
ビット操作命令のマシン語 ………………138
ビット番号 …………………………………210
比較命令………………………………………64
比較命令のマシン語…………………………68
被除数 ………………………………………224
フェッチ ……………………………………260
ブランチ条件 ………………………………141
プリ・デクリメント・
　アドレス・レジスタ間接 …………………16
プログラム・カウンタ………………………10
プログラム制御命令…………………………31
符号拡張……………………………65,226
分岐命令 ……………………………………146
分岐命令のマシン語 ………………………155
変位(バイト)…………………………………55
ポスト・インクリメント・
　アドレス・レジスタ間接 …………………16
〈ま　行〉
マシン語………………………………………22
マシン語フォーマット………………………23
マルチ・プロセッサ・システム……………88
無条件分岐命令 ……………………………146
メモリ内容 …………………………………244
メモリ内容をシフト ………………………104
〈や　行〉
ユーザースタック・ポインタ(USP)………12
ユーザーバイト………………………………13
ユーザーフラグ………………………………13
優先度 ………………………………………260
〈ら　行〉
リセット信号 ………………………………244
リセット例外処理 …………………………260
リターン命令のマシン語 …………………166
レジスタ間接…………………………………17
レジスタ構成…………………………………10
レジスタ内容をシフト ……………………104
レジスタ・フィールド ……………………104
レジスタ・リスト …………………………236
例外処理……………………………58,184
例外処理要因 ………………………………260
例外ベクタ・アドレス ……………………184
ローテート命令のマシン語 ………………122
ロング即値……………………………………42
ロング・ワード………………………………35
ロング・ワード・オペレーション…………23
ロング・ワード・レジスタ…………………10
論理演算命令…………………………………31
論理演算命令のマシン語……………………82
論理積命令……………………………………75
論理和命令……………………………………77
〈わ　行〉
ワード…………………………………………35
ワード・オペレーション……………………23
ワード・レジスタ……………………………10
ワード即値……………………………………42
割込マスク……………………………………14
割込例外処理 ………………………………260

アルファベット順

ABCD命令……………………………95,192
ADD命令 …………………………40,192
ADDA命令 ………………………41,192
ADDI命令 ………………………42,194
ADDQ命令 ………………………43,194
ADDX命令 ………………………43,194
AND命令 …………………………75,196
ANDI命令 ………………………76,196
ASL命令 ……………………106,196,198
ASR命令 ……………………………108,198
Bcc命令 ……………………………149,200
BCC命令 ……………………………149,200
BCD演算命令 ……………………………95
BCD演算命令のマシン語 …………………99
BCHG命令 ……………………………137,208
BCLR命令 ……………………………135,206
BCS命令 ……………………………149,202
BEQ命令 ……………………………149,202
BGE命令 ……………………………149,202
BGT命令 ……………………………150,202
BHI命令 ……………………………150,202
BLE命令 ……………………………151,204
BLS命令 ……………………………151,204
BLT命令 ……………………………152,204
BMI命令 ……………………………152,204
BNE命令 ……………………………152,204
BPL命令 ……………………………152,208
BRA命令 ……………………………148,206
BSET命令 ……………………………134,206
BSR命令 ……………………162,163,210
BTST命令 ……………………………132,210
BVC命令 ……………………………152,208
BVS命令 ……………………………152,208
Cフラグ …………………………………57
CCR ……………………………………234
CHK命令……………………………189,210
CLR命令 ……………………………73,212
CMP命令 ……………………………64,212
CMPA命令 …………………………65,212
CMPI命令……………………………65,212
CMPM命令 …………………………67,214
CPU制御命令 ……………………………31
DBcc命令……………………………153,214
DBCC命令 …………………………155,216
DBCS命令 …………………………155,216
DBEQ命令 …………………………155,218
DBF命令 ……………………………155,218
DBGE命令 …………………………155,218
DBGT命令 …………………………155,218
DBHI命令 …………………………155,218
DBLE命令 …………………………155,220
DBLS命令 …………………………155,220
DBLT命令 …………………………155,220
DBMI命令 …………………………155,220
DBNE命令 …………………………155,220
DBPL命令 …………………………155,222

DBRA命令 ……155,222
DBT命令 ……155,222
DBVC命令 ……155,222
DBVS命令 ……155,222
DISP 8 ……164
DISP16 ……164
DIVS命令 ……58,224
DIVU命令 ……58,224
EOR命令 ……79,226
EORI命令 ……80,226
Exception Processing ……184
EXG命令 ……36,226
EXT命令 ……226
i/rフィールド ……104
JMP命令 ……146,228
JSR命令 ……162,228
LEA命令 ……36,228
LINK命令 ……36,173,174
LINK命令の使い方 ……177
LINK命令のマシン語 ……179
LSL命令 ……102,230
LSR命令 ……104,230
MOVE命令 ……35,232
MOVEA命令 ……36,234
MOVE from SR ……234
MOVEM命令 ……36,236
MOVEP命令 ……36,238
MOVEQ命令 ……36,238
MULS命令 ……57,238
MOVE to CCR命令 ……234
MOVE to/from USP命令 ……234
MOVE to SR命令 ……234
MULU命令 ……56,240
Nフラグ ……59
NBCD命令 ……98,240
NEG命令 ……240
NEGX命令 ……240
NOP命令 ……242
NOT命令 ……80,242
OPモード・フィールド ……41
Operating System Call ……183
OR命令 ……77,242
ORI命令 ……78,242
PEA命令 ……37,244
RESET命令 ……35,244
R/Mビット ……44
ROL命令 ……114,244
ROR命令 ……116,246
ROXL命令 ……118,246
ROXR命令 ……120,248
RTE命令 ……250
RTR命令 ……165,250
RTS命令 ……164,250
SBCD命令 ……97,250
Scc命令 ……35,252
SCC命令 ……254
SCS命令 ……254
SEQ命令 ……254
SF命令 ……254
SGE命令 ……254
SGT命令 ……256
SHI命令 ……256
SLE命令 ……256
SLS命令 ……256
SLT命令 ……256
SMI命令 ……258
SNE命令 ……258
SPL命令 ……258
ST命令 ……258
STOP命令 ……35,260
SUB命令 ……45
SUBA命令 ……45,262
SUBI命令 ……46,262
SUBQ命令 ……46,262
SUBX命令 ……46,264
SVC命令 ……258
SVS命令 ……260
SWAP命令 ……36,264
Tビット ……260
TAS命令 ……89,264
TRAP命令 ……183,184,264
TRAPV命令 ……188,266
TST命令 ……74,266
UNLK命令 ……36,173,175,266
UNLK命令の使い方 ……177
UNLK命令のマシン語 ……179
V30 ……4
Vフラグ ……57
Zフラグ ……59

数　字

68000 ……3
68000と8086の命令比較 ……32
70116 ……4
80186 ……3
80286 ……3
8086 ……3
8088 ……3

《著者略歴》

村山　仁郎(むらやま　よしろう)

昭和47年3月／早稲田大学大学院理工学研究科電気工学専攻修士課程卒業

昭和48年／第一システム㈱設立

現在＝第一システム㈱代表取締役

主な著書＝マイクロコンピュータ16ビットプログラミング技法(産報出版)，マイクロコンピュータ実用モニタとアセンブラ／8086マシン語プログラミング(秋葉出版)

工学選書 2

68000マシン語プログラミング

定価2000円

昭和61年3月10日　初版発行

著　者　　村山仁郎

発行者　　星　正明

発行所　　株式会社工学社

〒151　東京都渋谷区代々木1-37-1ぜんらくビル

電　話　(03)375-5784代〔営業〕

(03)320-1218代〔編集〕

振替口座　東京5-22510

印刷：凸版印刷株式会社

企画・編集：有限会社ソレカラ社

表紙CG製作：株式会社アルファ・ベータ

ISBN4-87593-076-3 C3055 ¥2000E

工学選書シリーズ

1	C言語と周辺制御	新津 靖/池上皓三 著
2	68000マシン語プログラミング	村山仁郎 著

定価2000円

ISBN4-87593-076-3 C3055 ¥2000E